MEASURE BEFORE YOU CODE: TOOL-AUGMENTED CAD PROGRAM SYNTHESIS FROM POINT CLOUDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Synthesizing executable CAD programs from point clouds requires recovering both discrete modeling sequences and continuous geometric parameters. Recent point-to-CAD methods rely on multimodal large language models (MLLMs) to compress point clouds into latent tokens and infer parametric programs end-to-end. While these methods show impressive progress in recovering overall shape structure, this paradigm overlooks the primary advantage of point clouds over 2D images: direct access to explicit 3D coordinates. Dimensions are therefore predicted rather than measured, which makes them prone to error. We propose QM-CAD (Query-and-Measure CAD), an agentic framework guided by a simple principle: measuring before coding. An MLLM drafts a modeling plan that directs geometric queries to the raw point cloud. A query-driven segmenter identifies the surface points relevant to each planned operation. A geometry fitter returns measured parameters and point projections. The MLLM uses this numerical and visual feedback to generate the final CAD program. To support this paradigm, we construct an automated pipeline for synthesizing plan-measure-generate trajectories with real tool executions. Supervised fine-tuning on these trajectories teaches the model to formulate queries and use their feedback. Reinforcement learning then refines the model, using the solid Intersection-over-Union (IoU) of executed CAD models as the reward. Experiments on DeepCAD and Fusion360 show that QM-CAD achieves higher geometric reconstruction accuracy than recent point-to-CAD methods while keeping a low invalidity ratio.

1 INTRODUCTION

Point-to-CAD reverse engineering recovers an executable, editable CAD program from an unstructured set of 3D surface points (Rukhovich et al., 2025). The program consists of a discrete sequence of modeling operations, each with continuous geometric parameters (Wu et al., 2021; Willis et al., 2021). Recent work unifies text, image, and point-cloud inputs under one encode-and-generate architecture. The input is encoded into latent tokens, from which a large language model (LLM) directly generates the CAD program, either as a command sequence (Xu et al., 2024a) or as executable code (Kolodiazhnyi et al., 2025).

For reconstruction, the key difference among the three inputs is how much 3D information each retains (Kolodiazhnyi et al., 2025). LLM-based CAD generation began with text (Badagabettu et al., 2024; Wang et al., 2025a; Guan et al., 2025), which retains the least: a prompt carries only what it states, and critical dimensions can be missing (Yuan et al., 2026). As multimodal LLMs (MLLMs) matured, work split into two branches that reconstruct CAD from images (Wang et al., 2025b; Elistratov et al., 2026) and from point clouds (Rukhovich et al., 2025). Images retain more, but not all: projection loses depth and occlusion hides surfaces (Xu et al., 2024b), and multiple views recover part of this only after they are brought into correspondence (Hartley & Zisserman, 2004). Point clouds retain the most: they give the 3D coordinates of the surface directly (Xu et al., 2024b). Point clouds should therefore yield the most accurate reconstructions, but reported results show otherwise. On the standard DeepCAD and Fusion360 benchmarks, the same method achieves nearly the same accuracy from point clouds as from multi-view images (Kolodiazhnyi et al., 2025; Kabisov et al., 2026a), and the best point-cloud and image results are on par (Rukhovich et al., 2025; Elistratov et al., 2026). These observations raise a question: *Why does the input that retains the most 3D information fail to yield the most accurate reconstructions?*

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.

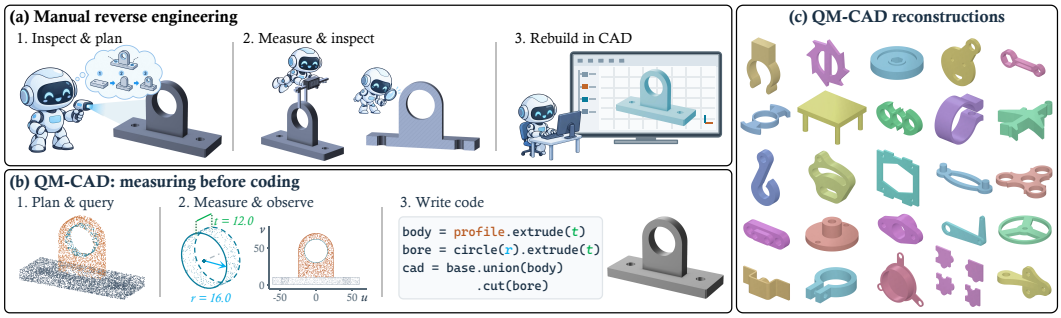


Figure 1: **Measuring before coding.** (a) In manual reverse engineering, an engineer inspects the part and plans its components, measures each one and inspects its cross-section, and rebuilds it in CAD. (b) QM-CAD follows the same workflow on a point cloud: it plans the construction and queries each step, measures parameters and observes a projection of the step points, and writes the CAD program from these measurements. (c) Reconstructions by QM-CAD.

We attribute this gap to how existing methods represent point clouds: a point encoder compresses them into latent tokens, and the LLM generates the program from these tokens alone. This representation causes two problems. (1) The MLLM has no prior for point tokens. Images and text benefit from priors learned in large-scale pre-training, whereas paired point-cloud and language data are scarce, so point clouds must be aligned from scratch with an extra encoder and limited data (Xu et al., 2024b; Qi et al., 2024; Tang et al., 2025). This lack of priors offsets the extra 3D information that point clouds retain. (2) Latent tokens hide the exact coordinates, so continuous CAD parameters must be regressed from the latent space. This regression exposes a known weakness of LLMs: they excel at discrete structure and sequences, not at precise continuous values. Their default number encodings are discrete and discontinuous (Golkar et al., 2023), they lack an inductive bias for emitting numbers (Zausinger et al., 2025), and they frequently fail even at basic numerical tasks such as comparing decimals (Yang et al., 2025b). In CAD, prior work therefore discretizes continuous parameters (Wu et al., 2021; Wang et al., 2025b); in general LLM reasoning, exact computation is delegated to external programs (Gao et al., 2023; Chen et al., 2023). Yet point clouds already carry these values: encoding them only to recover them later is a detour that turns directly measurable dimensions into prediction targets. Both problems thus stem from the representation, not from the point cloud itself. *So, how can we present point clouds in forms that the MLLM already understands, while keeping the dimensions of the part readable?*

We find an answer in how human engineers reverse engineer a part. An engineer first inspects the whole part, identifies its components and the order in which to model them, and only then measures each component with calipers. For a shape extruded from a complex sketch, the engineer cuts the part open and inspects the cross-section directly. Inspired by this workflow, we propose QM-CAD (Query-and-Measure CAD), which follows one principle: **measuring before coding** (Figure 1). Given the point cloud, an MLLM first drafts a modeling plan that lists the parts to construct. For each planned step, it then issues a geometric query to the raw point cloud: measurement tools locate the surface points of that step, measure parameters such as radius, depth, and position, and return a projection of these points onto the sketch plane for sketch-extrude structures. Finally, the MLLM generates the complete CadQuery program from the plan, the measured parameters, and the projections. *This design addresses both problems.* The measurements arrive as text and the projections as images, drawn like engineering drawings with coordinate axes and numeric ticks; these are the two modalities for which the MLLM already has strong pre-trained priors, and through tool feedback the point-cloud task borrows those priors. Dimensions are grounded in these measurements and scaled projections, rather than relying solely on latent features. To train this workflow, we automatically construct a dataset of complete plan-measure-generate trajectories from executable CAD programs, with every measurement produced by real tool execution. After supervised fine-tuning on these trajectories, reinforcement learning optimizes planning, querying, and code generation together, with the solid IoU of the executed program as the reward.

On DeepCAD and Fusion360, QM-CAD achieves higher geometric reconstruction accuracy while keeping a low invalidity ratio. Our contributions are threefold:

- **Measuring before coding for point-to-CAD.** A modeling plan directs geometric queries to the raw point cloud and associates each construction step with its surface points. The tool feedback reaches the MLLM as text and images, the modalities it was pre-trained on, and grounds the dimensions in measurements.
- **An automated plan-measure-generate trajectory synthesis pipeline.** From executable CAD programs, we derive modeling plans, step surface masks, and real tool feedback, and assemble them into complete training trajectories.
- **Measurement-in-the-loop policy optimization.** With the solid Intersection-over-Union (IoU) of executed CAD models as the reward, we jointly optimize planning, querying, and code generation within one trajectory.

2 Related Work

2.1 CAD Reconstruction from Point Clouds

Learning-based point-to-CAD methods predict modeling operations and their parameters from point representations. Sequence-based approaches map point clouds into the latent space of sketch-extrude autoencoders (Wu et al., 2021), align geometry and construction sequences through contrastive learning (Ma et al., 2023), or recover command and parameter tokens through masked-token diffusion (Ma et al., 2024). Structured decoders introduce step-conditioned attention (Khan et al., 2024a), hierarchical loop and extrusion decoding (Dupont et al., 2024), and direct sketch-primitive prediction (Ma et al., 2025). Related generators condition CAD synthesis on text, images, or multiple modalities (Khan et al., 2024b; Chen et al., 2025; Xu et al., 2024a). Code-based methods replace specialized command vocabularies with executable scripts, drawing on the program priors of pre-trained LLMs. CAD-Recode maps point embeddings to CadQuery programs (Rukhovich et al., 2025), cadrille supports multimodal inputs (Kolodiazhnyi et al., 2025), and CAD-Coder combines text-to-code generation with chain-of-thought planning (Guan et al., 2025). All of these predict dimensional values from learned features.

A geometry-first line instead estimates CAD parameters directly from the input geometry through segmentation, fitting, and optimization. SPFN fits primitives from predicted point memberships and normals (Li et al., 2019), Point2Cyl derives extrusion axes, ranges, and sketches from segmented base and barrel points (Uy et al., 2022), Point2CAD reconstructs B-Rep geometry by fitting and intersecting surface patches (Liu et al., 2024), and MiCADangelo recovers constrained sketches from rasterized mesh cross-sections and optimizes extrusion lengths against the scan (Karadeniz et al., 2025). These methods ground parameters in explicit geometry, but within a fixed pipeline rather than a construction plan. QM-CAD draws on both lines: an LLM plans the construction from learned point features, and segmentation and fitting serve as measurement tools whose targets the plan specifies. CAD-SIGNet already directs attention to points of the current modeling step; QM-CAD makes this association an explicit step surface mask and measures from the selected points.

2.2 Feedback and Tool Use in CAD Generation

Feedback improves CAD generation through training rewards and iterative reconstruction. CAD-Fusion uses visual feedback during training (Wang et al., 2025a), while CAD-Coder and cadrille optimize generation with rewards from executed geometry (Guan et al., 2025; Kolodiazhnyi et al., 2025). CADReasoner edits programs and CADENA extends them step by step using comparisons between current and target geometry (Kabisov et al., 2026a;b). PS-CAD uses residual point clouds and planar prompts to generate and select candidate operations (Yang et al., 2025a). CADFit fits parametric operations to the target geometry and assembles the program by IoU-driven selection and residual refinement (Nehme et al., 2026). QM-CAD queries the surface points associated with a planned operation to obtain measured parameters before final code generation.

Tool-augmented CAD agents also acquire geometric observations during generation. CAD-Assistant supplies rendering, sketch parameterization, and cross-section extraction tools (Mallis et al., 2025), while TOOLCAD learns interactions with a CAD engine through reinforcement learning (Gong et al., 2026). ProCAD (Yuan et al., 2026) asks the user for missing specifications before code synthesis. QM-CAD specializes tool use for point-to-CAD through plan-conditioned surface

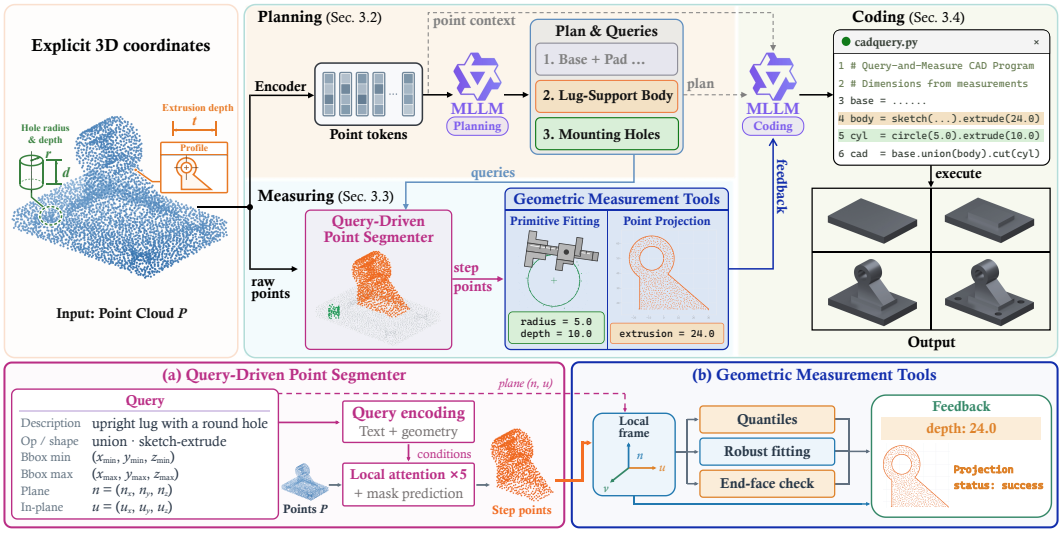


Figure 2: **Overview of QM-CAD.** The essential idea is *measuring before coding*. Rather than regressing dimensions from point embeddings, QM-CAD plans the construction first and then measures each planned step on the raw points. The measurements return as structured text and projection images, the two modalities in which the MLLM already has strong priors, so the program parameters are grounded in measurements and projections. **(a)** The query of each step conditions the segmenter, which selects the step points P_k from the raw point cloud. **(b)** The tools measure P_k in the local frame of the query plane and return parameters, a status, and a projection; a field without enough supporting points is reported as unknown.

segmentation and measurement. Program-derived step surface masks supervise segmentation, and trajectories with real tool executions teach the generator to use the returned parameters and projections. Its measurement-in-the-loop policy optimization includes these calls and observations in the rollouts, connecting execution-based rewards to both planning and the use of measurements.

2.3 Language Models, Numbers, and Tools

Language models handle discrete tokens well but continuous quantities poorly. Their number encodings are discrete and discontinuous (Golkar et al., 2023), cross-entropy training provides no notion of numeric proximity (Zausinger et al., 2025), and even strong models fail at basic numerical tasks (Yang et al., 2025b). One remedy is to let the model reason and delegate computation: PAL and Program of Thoughts generate programs that an interpreter executes (Gao et al., 2023; Chen et al., 2023), and HuggingGPT, ViperGPT, and VisProg dispatch subtasks to neural tools (Shen et al., 2023; Surís et al., 2023; Gupta & Kembhavi, 2023). Language-driven segmentation makes a mask decoder itself the tool: LISA, SegPoint, and Reason3D emit segmentation tokens whose embeddings a decoder turns into 2D or 3D masks (Lai et al., 2024; He et al., 2024; Huang et al., 2025), and PQ3D unifies 3D vision-language tasks through promptable queries (Zhu et al., 2024). Point-cloud LLMs align a point encoder with the language model through paired data (Xu et al., 2024b; Qi et al., 2024), which remains scarce (Tang et al., 2025). QM-CAD applies this delegation principle to point-to-CAD reverse engineering. Unlike reasoning segmentation, where the mask is the final output, the mask here is an intermediate result that serves measurement.

3 Method

3.1 Overview

QM-CAD applies the principle of measuring before coding: it plans the construction, measures each planned step on the raw point cloud, and writes the program from both (Figure 2). The input is a surface point cloud $P = \{p_i\}_{i=1}^{N}$ with $p_i \in \mathbb{R}^3$, centered and scaled in the same way as the

training data (Appendix A.1). Tool feedback and generated programs share one coordinate frame, so measured values can be written into the program directly.

The policy and the tools use P in different forms. The policy π_θ, a Qwen3.5-2B multimodal model (Qwen Team, 2026) with parameters θ, sees the whole shape through point tokens produced by a Point-PQAE encoder (Zhang et al., 2025a) and a learned projector. The measurement tools read the exact coordinates of P. Generation proceeds in two turns:

$$A \sim \pi_\theta(\cdot \mid P), \qquad F = \mathcal{T}_\phi(P, A), \qquad C \sim \pi_\theta(\cdot \mid P, A, F). \tag{1}$$

In the planning turn, the policy outputs A: a plan of K modeling steps and one tool call that holds a geometric query q_k for each step k. The tool system $\mathcal{T}_\phi$, consisting of the query-driven point segmenter and the geometric measurement tools, answers with feedback $F = \{F_k\}_{k=1}^{K}$; its learned segmenter has parameters ϕ. In the code turn, the policy writes the CadQuery program C. The policy thus generates A and C, whereas F is measured on the raw point cloud.

3.2 Planning the Construction

The planning turn decides what to build before anything is measured. It decomposes the part into an ordered sequence of K steps, each a complete operation entity, either a primitive or an extruded sketch, together with the Boolean operation that attaches it to the part; for the bracket in Figure 2, these are the base with its pad, the lug-support body, and the mounting holes. For each step, the policy issues a query q_k that specifies the operation, the shape type, a coarse bounding box, a short shape description, and the plane orientation (XY, YZ, or ZX). The bounding boxes in the plan only need to localize each step, because the tools supply the measurable dimensions; QDPS and the policy are trained on randomly perturbed boxes (Appendix C.4). The queries of all K steps are answered in one tool call (Appendix A.3), so the code turn is conditioned on the measurements of every step at once, including steps that come later in the plan.

3.3 Measuring Each Step

Locating the step. A Query-Driven Point Segmenter (QDPS; Figure 2a) encodes the query into a query feature that modulates the point features in a serialized local-attention network (Appendix B.1). A pointwise head then predicts a membership logit $f_\phi(P, q_k)_i$ for every point p_i, which gives the membership probabilities and the selected subset

$$\hat{m}_{ki} = \sigma(f_\phi(P, q_k)_i), \qquad P_k = \{p_i \in P : \hat{m}_{ki} \geq \eta\}, \tag{2}$$

where $\hat{m}_{ki}$ is the probability that p_i lies on the surface of step k, σ is the sigmoid function, and η is the mask threshold. Because each query is predicted independently, a surface point shared by several operations can belong to all of their subsets.

Measuring the step. A Statistical Geometry Fitter (SGF; Figure 2b) expresses P_k in a local frame aligned with the query plane, estimates extents from coordinate quantiles, and estimates centers and radii by robust primitive fitting. The end positions and depth of an extrusion are returned only when enough points lie on both end faces; otherwise they are reported as unknown rather than guessed. Because numbers alone cannot describe a sketch profile, SGF also projects the point cloud onto the query plane, highlights the points of P_k, and renders the projection with equal-scale axes and numeric ticks (Appendix B). The response to q_k is $F_k = (g_k, I_k, s_k)$, consisting of measured parameters g_k, a projection image I_k for sketch-extrude steps, and a success, partial, or failure status s_k. The parameters and status return as structured text and the projection as an image, the two modalities in which the policy already has strong pre-trained priors.

3.4 Coding the Program

With the plan and the feedback in context, the code turn writes the complete program C (Figure 2, top right). The plan determines the sequence of operations; measured parameters are written from their values, and parameters that the step points alone cannot determine, such as those of faces hidden in the final shape, are completed from related measurements of other steps, the projections, and the point tokens. The policy thus handles the discrete structure, while the continuous values are grounded in measurements of the raw point cloud. Both turns are learned from trajectories, which we construct automatically as follows.

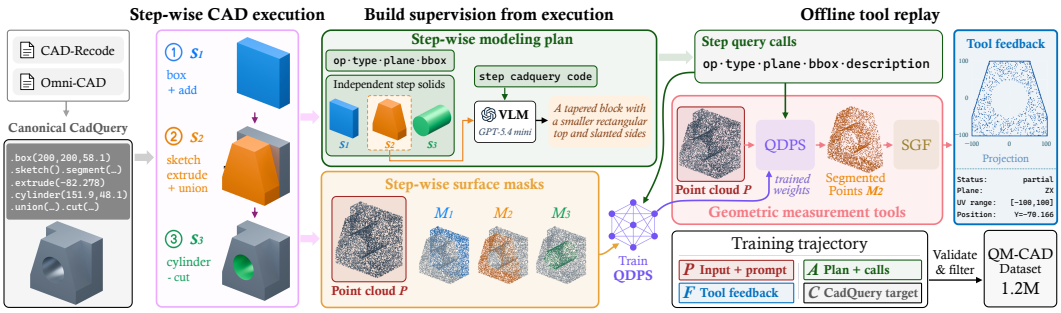


Figure 3: **Construction of the QM-CAD training data.** Top, from left to right: step-wise execution of a source program, supervision derived from the executed steps, and offline tool replay with SGF and the trained QDPS. Steps S_1–S_3 keep the same colors throughout the figure. Bottom: the four components of one resulting training trajectory.

3.5 Training Data Construction

Training requires trajectories (P, A, F, C) for the policy and step surface masks for QDPS. We synthesize both automatically from executable CAD programs (Figure 3); the resulting trajectories form the dataset $\mathcal{D}$ (statistics in Appendix C.5).

Programs, plans, and masks. We convert CAD-Recode (Rukhovich et al., 2025) programs and Omni-CAD (Xu et al., 2024a) command sequences into a common CadQuery style (Appendix C). Executing each program step by step yields its operation entities and modeling plan. A vision-language model writes the shape description of each sketch-extrude step from images and code of that step (Appendix C.3). The same execution supervises QDPS: the final-surface points within a geometric tolerance of a step's operation entity form its step surface mask (Eq. (8), Appendix C.2), the target for $\hat{m}_{ki}$ in Eq. (2).

Tool replay. The feedback in each trajectory is produced by running QDPS and SGF on the input point cloud rather than copied from ground truth. Partial and failed responses are kept: a face hidden in the final shape cannot be measured directly, but its value usually follows from the context, for example from the measured plane of another step that the face lies on. The policy is therefore trained on trajectories in which such parameters must be completed from the context.

3.6 Training

The point encoder is first adapted to CAD surfaces by point reconstruction, and the projector is then aligned with the language model on Text2CAD (Khan et al., 2024b) samples (Appendix D).

Segmentation supervision. QDPS learns the probabilities $\hat{m}_{ki}$ in Eq. (2) from the step surface masks, with a false-positive penalty that suppresses unrelated surfaces, which would otherwise contaminate measurements (Eq. (7), Appendix B.1). QDPS and SGF then stay fixed during policy training, so the policy trains with the same tools it uses at inference.

Supervised fine-tuning. The policy is first trained on $\mathcal{D}$ by supervised fine-tuning (SFT) on the two quantities it generates in Eq. (1):

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(P,A,F,C)\sim\mathcal{D}}\left[\log \pi_\theta(A \mid P) + \log \pi_\theta(C \mid P, A, F)\right], \tag{3}$$

where each log-probability sums over the corresponding output tokens. Because Eq. (3) uses F only as context, the policy learns which queries to issue and how to use their answers, without being trained to predict tool outputs.

Measurement-in-the-loop policy optimization. SFT imitates reference trajectories, but it cannot tell the policy whether its own queries lead to accurate geometry. Reinforcement learning (RL)

supplies this signal. RL uses a subset of the training sets of DeepCAD (Wu et al., 2021) and Fusion360 (Willis et al., 2021). For each input, the policy samples G candidate trajectories; candidate i issues its own queries, receives real tool feedback, and writes a program C_i. With $S_i = \mathrm{Exec}(C_i)$ the solid obtained by executing C_i and $S^\star$ the target solid, the reward is

$$R_i = \begin{cases} \mathrm{IoU}_{\mathrm{solid}}(S_i, S^\star), & \text{if the candidate is valid,} \\ -1, & \text{otherwise,} \end{cases} \tag{4}$$

where a candidate is valid if its program executes into a valid solid, and $\mathrm{IoU}_{\mathrm{solid}}$ is the volumetric intersection over union of the two solids (Eq. (13), Appendix E.2). The rewards in Eq. (4) are compared within each group to form advantages, and the policy is updated with a GSPO-style sequence-level objective (Zheng et al., 2025) whose importance ratio covers the tokens of both turns (Appendix E). The shared advantage applies to both turns, so planning and querying are updated together with code generation from the same reward.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on the official test sets of DeepCAD (Wu et al., 2021) and Fusion360 (Willis et al., 2021), the two standard benchmarks for point-to-CAD reconstruction.

Metrics. Following prior work (Rukhovich et al., 2025; Kolodiazhnyi et al., 2025), we report three metrics, computed with the official evaluation script of cadrille: Chamfer distance (CD), Intersection over Union (IoU), and invalidity ratio (IR). Because invalid predictions bias the mean, we report the median CD, multiplied by 10^3. IoU is computed between the reconstructed and ground-truth solids and averaged over the test set, in percent. IR is the percentage of generated programs that fail to produce a valid CAD model.

Baselines. We compare with methods that generate a CAD command sequence or CAD code from a point cloud: CAD-SIGNet (Khan et al., 2024a), CAD-Recode (Rukhovich et al., 2025), cadrille (Kolodiazhnyi et al., 2025), and CADReasoner (Kabisov et al., 2026a). All methods are scored on a single sample per shape. CAD-Recode, cadrille, and CADReasoner use their official inference configurations, and the CAD-SIGNet results without test-time sampling are taken from cadrille (Kolodiazhnyi et al., 2025). For CADReasoner, which reports the best of its t programs under the evaluation metric, we run the released model for $t = 5$ iterations and score the program of the final iteration (Appendix F).

Implementation details. QM-CAD uses Qwen3.5-2B as the policy and follows the inference procedure of Section 3. Model, tool, and training configurations are given in Appendices A–E.

4.2 Comparison with Prior Methods

Quantitative results. Table 1 compares QM-CAD with prior methods on DeepCAD and Fusion360. QM-CAD achieves the best CD and IoU on both benchmarks under the single-output protocol used in the table. Against the strongest baseline under this protocol, it lowers the median CD by 10% on DeepCAD (0.174 to 0.156) and by 12% on Fusion360 (0.173 to 0.152), and raises the IoU by 4.1 and 2.4 points. QM-CAD improves geometric reconstruction accuracy while keeping a low invalidity ratio. The ablation below traces most of the gain to measuring before coding.

Qualitative results. Figure 4 compares reconstructions on DeepCAD and Fusion360. On DeepCAD, the baselines recover the overall shape but miss or distort small features, such as the bolt holes of the disc, the holes of the lever, and the spokes of the wheel; QM-CAD recovers all three. Such features are small relative to the part, and QM-CAD grounds their positions and sizes in measurements and projections of the step points rather than relying solely on latent tokens. On the more complex Fusion360 parts, the baselines often split a part into disconnected pieces or lose thin members, whereas QM-CAD keeps each part connected and preserves thin members such as the two parallel bars and the prongs (more examples in Appendix H).

Table 1: **Point-to-CAD reconstruction on DeepCAD and Fusion360.** CD is the median Chamfer distance ($\times 10^3$), IoU is the mean Intersection over Union (%), and IR is the invalidity ratio (%). Best results are in bold, second-best are underlined. †Results without test-time sampling, as reported by cadrille. CADReasoner is scored at the final iteration ($t = 5$); see Appendix F.

	DeepCAD			Fusion360		
Method	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓
CAD-SIGNet† (Khan et al., 2024a)	0.290	77.3	5.00	0.700	58.4	9.30
CAD-Recode (Rukhovich et al., 2025)	0.177	<u>91.2</u>	3.33	0.184	82.1	4.93
cadrille (Kolodiazhnyi et al., 2025)	<u>0.174</u>	91.0	**0.10**	<u>0.173</u>	<u>85.2</u>	<u>0.23</u>
CADReasoner (Kabisov et al., 2026a)	0.194	89.0	1.72	0.216	81.7	2.67
QM-CAD (ours)	**0.156**	**95.3**	**0.10**	**0.152**	**87.6**	**0.17**

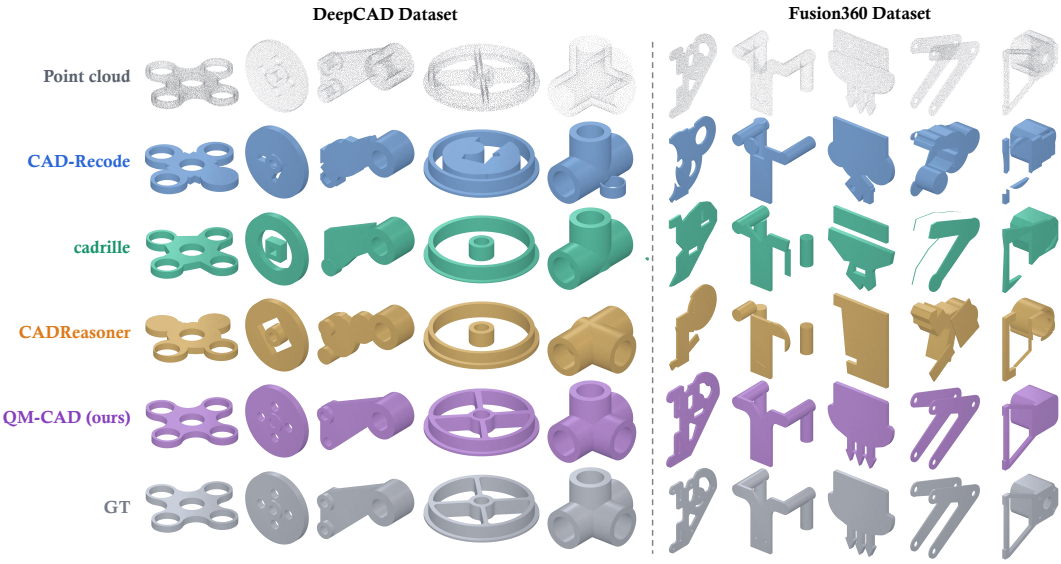


Figure 4: **Qualitative comparison.** Columns show parts from DeepCAD (Wu et al., 2021) (left) and Fusion360 (Willis et al., 2021) (right). Rows from top to bottom: input point cloud, CAD-Recode (Rukhovich et al., 2025), cadrille (Kolodiazhnyi et al., 2025), CADReasoner (Kabisov et al., 2026a), QM-CAD, and the ground truth.

4.3 Analysis

Ablation study. We ablate the components that QM-CAD adds to a direct point-to-code generator (Table 2). *w/o planning and tools* removes the planning turn and the measurement tools altogether: the policy writes the program directly from the point tokens, as in the encode-and-generate paradigm. *w/o tools* keeps the planning turn but removes the tool calls, so the program is written from the plan and the point tokens without measurements. Both variants are trained with the same SFT and RL configurations as QM-CAD. *w/o RL* keeps the full query-and-measure pipeline but stops after SFT, which isolates the effect of RL.

Measuring before coding brings most of the gain over prior methods. The *w/o planning and tools* variant, a direct generator that shares the backbone, data, and training with QM-CAD, performs on par with the strongest baselines, for example 91.3 IoU on DeepCAD against 91.2 for CAD-Recode. Adding planning and tools raises the IoU by 4.0 points on DeepCAD and 2.1 points on Fusion360, and lowers CD and the invalidity ratio on both benchmarks. Planning without measurements changes little: *w/o tools* stays close to the direct generator, with an IoU of 91.9 on DeepCAD and 85.7 on Fusion360. The gain appears once the tools answer the queries of the plan, which raises the IoU by a further 3.4 and 1.9 points and lowers CD and the invalidity ratio on both benchmarks. Reinforcement learning mainly improves validity. With SFT alone, the query-and-measure policy is already more

Table 2: **Ablation study.** All variants share the same backbone and training data. Metrics as in Table 1; best results in bold.

	DeepCAD			Fusion360		
Variant	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓
w/o planning and tools	0.169	91.3	0.17	0.167	85.5	0.23
w/o tools	0.171	91.9	0.15	0.163	85.7	0.29
w/o RL	0.161	93.7	0.44	0.158	85.9	2.38
QM-CAD (full)	**0.156**	**95.3**	**0.10**	**0.152**	**87.6**	**0.17**

Table 3: **Accuracy of the measurement tools on the DeepCAD test set.** Each step uses its reference query. Segmentation metrics are averaged over steps. Coverage is the fraction of observable parameters that SGF returns, and Acc@τ is the fraction of observable parameters returned with an error of at most τ of the longest axis L of the part; the error of a vector parameter is its largest component error. Measurement metrics are pooled over all parameter types (Table 12, Appendix G). All values are in %.

Segmentation (QDPS)			Measurement (QDPS + SGF)			
IoU↑	Precision↑	Recall↑	Coverage↑	Acc@0.1%↑	Acc@0.5%↑	Acc@1%↑
94.6	96.2	98.1	96.5	90.2	94.4	95.0

accurate than the direct generator on both benchmarks, but more of its programs fail to execute, with invalidity ratios of 0.44 and 2.38. RL, whose reward is -1 for invalid candidates, lowers these ratios to 0.10 and 0.17 and further improves CD and IoU.

Measurement accuracy. Because the dimensions in the program are grounded in tool measurements, the tools must select the right points and measure them accurately. We evaluate them in isolation on the DeepCAD test set: each step is issued with its reference query, QDPS predicts the step mask, and SGF measures the selected points. The predicted masks are compared with the step surface masks (Eq. (8), Appendix C.2), and each returned parameter with its ground-truth value when that parameter is observable under the reference mask (Table 3; further statistics in Appendix G). QDPS reaches 96.2% precision and 98.1% recall, so few points from other surfaces enter the measurements. SGF returns 96.5% of the observable parameters, 95.0% within 1% of L and 90.2% within 0.1% of L. The SGF response-status distributions with predicted masks are close to those with ground-truth masks (Appendix G).

5 Conclusion

We presented QM-CAD, an agentic framework for point-to-CAD generation driven by the principle of *measuring before coding*. QM-CAD plans the construction, queries the raw point cloud for each step, and writes the program from the returned measurements and projections. The program parameters are therefore grounded in measurements rather than in latent features alone. Supervised fine-tuning on synthesized trajectories teaches the policy which queries to issue and how to use their answers. Measurement-in-the-loop reinforcement learning then optimizes planning, querying, and code generation together. On DeepCAD and Fusion360, QM-CAD achieves higher geometric reconstruction accuracy while keeping a low invalidity ratio. The ablations show that measuring before coding improves geometric reconstruction, while reinforcement learning further improves the geometry and lowers the invalidity ratio.

Future work can extend QM-CAD to further operations such as lofts, sweeps, fillets, and chamfers, by adding training data for them and dedicated segmentation and fitting tools for their surfaces. The same agentic loop could also be scaled to CAD assemblies and interactive editing.

AI USE STATEMENT

We used generative AI tools to aid and polish the writing of this paper and to search for related work. We also used them to assist with research execution, including implementing and running experiments. In constructing the synthetic training data, a vision-language model generated the shape descriptions of sketch steps (Appendix C). We have reviewed all AI-assisted work and 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 only publicly available CAD datasets (DeepCAD, Fusion360, CAD-Recode, Omni-CAD, and Text2CAD) under their respective licenses, and involves no human subjects, personal data, or crowdsourced annotation. The shape descriptions in the training data are annotated by a vision-language model rather than by human annotators. We see no ethical concerns specific to this work.

REPRODUCIBILITY STATEMENT

Section 3 describes QM-CAD, and the appendices give its implementation details: the model and input representation (Appendix A), the segmentation and measurement tools (Appendix B), the construction and filtering of the training data (Appendix C), and the SFT and RL configurations (Appendices D and E). Section 4.1 and Appendix F give the evaluation protocol, and Appendix G gives the protocol of the tool evaluation. All data used for training and evaluation are drawn from public datasets. We will release the QM-CAD dataset and the trained models upon publication.

REFERENCES

Akshay Badagabettu, Sai Sravan Yarlagadda, and Amir Barati Farimani. Query2CAD: Generating CAD models using natural language queries. *arXiv preprint arXiv:2406.00144*, 2024. URL https://arxiv.org/abs/2406.00144.

Cheng Chen, Jiacheng Wei, Tianrun Chen, Chi Zhang, Xiaofeng Yang, Shangzhan Zhang, Bingchen Yang, Chuan-Sheng Foo, Guosheng Lin, Qixing Huang, and Fayao Liu. CADCrafter: Generating Computer-Aided Design Models from Unconstrained Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11073–11082, 2025. URL https://arxiv.org/abs/2504.04753.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2211.12588.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. TransCAD: A Hierarchical Transformer for CAD Sequence Inference from Point Clouds. In *Computer Vision – ECCV 2024*, pp. 19–36, 2024. doi: 10.1007/978-3-031-73030-6_2. URL https://arxiv.org/abs/2407.12702.

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. URL https://arxiv.org/abs/2602.16317.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 2023. URL https://arxiv.org/abs/2211.10435.

Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, Bruno Régaldo-Saint Blancard, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. xVal: A Continuous Numerical

Tokenization for Scientific Language Models. *arXiv preprint arXiv:2310.02989*, 2023. URL https://arxiv.org/abs/2310.02989.

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 *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23161–23188, 2026. doi: 10.18653/v1/2026.findings-acl.1160. URL https://arxiv.org/abs/2604.07960.

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. In *Advances in Neural Information Processing Systems*, volume 38, pp. 59765–59789, 2025. URL https://arxiv.org/abs/2505.19713.

Tanmay Gupta and Aniruddha Kembhavi. Visual Programming: Compositional Visual Reasoning without Training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14953–14962, 2023. doi: 10.1109/CVPR52729.2023.01436. URL https://arxiv.org/abs/2211.11559.

Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

Shuting He, Henghui Ding, Xudong Jiang, and Bihan Wen. SegPoint: Segment Any Point Cloud via Large Language Model. In *Computer Vision – ECCV 2024*, pp. 349–367. Springer, 2024. doi: 10.1007/978-3-031-72670-5_20. URL https://arxiv.org/abs/2407.13761.

Kuan-Chih Huang, Xiangtai Li, Lu Qi, Shuicheng Yan, and Ming-Hsuan Yang. Reason3D: Searching and Reasoning 3D Segmentation via Large Language Model. In *International Conference on 3D Vision (3DV)*, pp. 1177–1186, 2025. doi: 10.1109/3DV66043.2025.00112. URL https://arxiv.org/abs/2405.17427.

Soslan Kabisov, Vsevolod Kirichuk, Andrey Volkov, Marina Barannikov, Gennadiy Savrasov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. CADReasoner: Iterative Program Editing for CAD Reverse Engineering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, pp. 6143–6153, June 2026a. URL https://arxiv.org/abs/2603.29847.

Soslan Kabisov, Gennadiy Savrasov, Maksim Elistratov, Antonio Rodriguez, Daniil Ignatiev, Nikita Gavrilov, Rustam Uzdenov, Alexey I. Boyko, Igor Pasechnik, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. CADENA: Stepwise CAD Reverse Engineering. *arXiv preprint arXiv:2608.00799*, 2026b. URL https://arxiv.org/abs/2608.00799.

Ahmet Serdar Karadeniz, Dimitrios Mallis, Danila Rukhovich, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. MiCADangelo: Fine-Grained Reconstruction of Constrained CAD Models from 3D Scans. In *Advances in Neural Information Processing Systems*, volume 38, pp. 82830–82848, 2025. URL https://arxiv.org/abs/2510.23429.

Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-SIGNet: CAD Language Inference from Point Clouds using Layer-wise Sketch Instance Guided Attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4713–4722, 2024a. URL https://arxiv.org/abs/2402.17678. Highlight.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating Sequential CAD Designs from Beginner-to-Expert Level Text Prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7552–7579, 2024b. doi: 10.52202/079017-0242. URL https://arxiv.org/abs/2409.17106. Spotlight.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal CAD Reconstruction with Reinforcement Learning. *arXiv preprint arXiv:2505.22914*, 2025. URL https://arxiv.org/abs/2505.22914.

Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. LISA: Reasoning Segmentation via Large Language Model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9579–9589, 2024. URL https://arxiv.org/abs/2308.00692.

Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. Supervised Fitting of Geometric Primitives to 3D Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2652–2660, 2019. URL https://arxiv.org/abs/1811.08988.

Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2CAD: Reverse Engineering CAD Models from 3D Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3763–3772, 2024. URL https://arxiv.org/abs/2312.04962. Highlight.

Weijian Ma, Minyang Xu, Xueyang Li, and Xiangdong Zhou. MultiCAD: Contrastive Representation Learning for Multi-modal 3D Computer-Aided Design Models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 1766–1776, 2023. doi: 10.1145/3583780.3614982. URL https://doi.org/10.1145/3583780.3614982.

Weijian Ma, Shuaiqi Chen, Yunzhong Lou, Xueyang Li, and Xiangdong Zhou. Draw Step by Step: Reconstructing CAD Construction Sequences from Point Clouds via Multimodal Diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 27144–27153, 2024. doi: 10.1109/CVPR52733.2024.02564. URL https://doi.org/10.1109/CVPR52733.2024.02564.

Xinzhu Ma, Cheng Wang, Chen Tang, Bin Wang, Shixiang Tang, Yuan Meng, Yunhong Wang, and Di Huang. Point2Primitive: CAD Reconstruction from Point Cloud by Direct Primitive Prediction. *arXiv preprint arXiv:2505.02043*, 2025. URL https://arxiv.org/abs/2505.02043.

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7284–7294, 2025. doi: 10.1109/ICCV51701.2025.00684. URL https://arxiv.org/abs/2412.13810.

Ghadi Nehme, Eamon Whalen, and Faez Ahmed. CADFit: Precise Mesh-to-CAD Program Generation with Hybrid Optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2605.01171.

OpenAI. Introducing GPT-5.4 mini and nano, March 2026. URL https://openai.com/index/introducing-gpt-5-4-mini-and-nano/.

Zekun Qi, Runpei Dong, Shaochen Zhang, Haoran Geng, Chunrui Han, Zheng Ge, Li Yi, and Kaisheng Ma. ShapeLLM: Universal 3D Object Understanding for Embodied Interaction. In *Computer Vision – ECCV 2024*, pp. 214–238. Springer, 2024. doi: 10.1007/978-3-031-72775-7_13. URL https://arxiv.org/abs/2402.17766.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse Engineering CAD Code from Point Clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9801–9811, 2025. doi: 10.1109/ICCV51701.2025.00914. URL https://arxiv.org/abs/2412.14042.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. In *Advances in Neural Information Processing Systems*, volume 36, pp. 38154–38180. Curran Associates, Inc., 2023. URL https://arxiv.org/abs/2303.17580.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual Inference via Python Execution for Reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11854–11864, 2023. doi: 10.1109/ICCV51070.2023.01092. URL https://arxiv.org/abs/2303.08128.

Yuan Tang, Xu Han, Xianzhi Li, Qiao Yu, Jinfeng Xu, Yixue Hao, Long Hu, and Min Chen. More Text, Less Point: Towards 3D Data-Efficient Point-Language Understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 7284–7292, 2025. doi: 10.1609/aaai.v39i7.32783. URL https://arxiv.org/abs/2408.15966.

Mikaela Angelina Uy, Yen yu Chang, Minhyuk Sung, Purvi Goel, Joseph G. Lambourne, Tolga Birdal, and Leonidas J. Guibas. Point2Cyl: Reverse Engineering 3D Objects from Point Clouds to Extrusion Cylinders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11850–11860, 2022. URL https://arxiv.org/abs/2112.09329.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-CAD Generation Through Infusing Visual Feedback in Large Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 65326–65345. PMLR, 2025a. URL https://arxiv.org/abs/2501.19054.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. CAD-GPT: Synthesising CAD Construction Sequence with Spatial Reasoning-Enhanced Multimodal LLMs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 7880–7888, 2025b. doi: 10.1609/aaai.v39i8.32849. URL https://arxiv.org/abs/2412.19663.

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):54, 2021. doi: 10.1145/3450626.3459818. URL https://arxiv.org/abs/2010.02392.

Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A Deep Generative Network for Computer-Aided Design Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6772–6782, 2021. URL https://arxiv.org/abs/2105.09492.

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. URL https://arxiv.org/abs/2411.04954.

Runsen Xu, Xiaolong Wang, Tai Wang, Yilun Chen, Jiangmiao Pang, and Dahua Lin. PointLLM: Empowering Large Language Models to Understand Point Clouds. In *Computer Vision – ECCV 2024*, pp. 131–147. Springer, 2024b. doi: 10.1007/978-3-031-72698-9_8. URL https://arxiv.org/abs/2308.16911.

Bingchen Yang, Haiyong Jiang, Hao Pan, Guosheng Lin, Jun Xiao, and Peter Wonka. PS-CAD: Local Geometry Guidance via Prompting and Selection for CAD Reconstruction. *ACM Transactions on Graphics*, 44(3):28, 2025a. doi: 10.1145/3733595. URL https://arxiv.org/abs/2405.15188.

Haotong Yang, Yi Hu, Shijia Kang, Zhouchen Lin, and Muhan Zhang. Number Cookbook: Number Understanding of Language Models and How to Improve It. In *International Conference on Learning Representations*, 2025b. URL https://arxiv.org/abs/2411.03766.

Bo Yuan, Zelin Zhao, Petr Molodyk, Bin Hu, and Yongxin Chen. Clarify Before You Draw: Proactive Agents for Robust Text-to-CAD Generation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.03045.

Jonas Zausinger, Lars Pennig, Anamarija Kozina, Sean Sdahl, Julian Sikora, Adrian Dendorfer, Timofey Kuznetsov, Mohamad Hagog, Nina Wiedemann, Kacper Chlodny, Vincent Limbach, Anna Ketteler, Thorben Prein, Vishwa Mohan Singh, Michael Morris Danziger, and Jannis Born. Regress, Don't Guess: A Regression-like Loss on Number Tokens for Language Models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 73995–74017. PMLR, 2025. URL https://arxiv.org/abs/2411.02083.

Xiangdong Zhang, Shaofeng Zhang, and Junchi Yan. Towards More Diverse and Challenging Pre-training for Point Cloud Learning: Self-Supervised Cross Reconstruction with Decoupled Views. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 28696–28706, 2025a. URL https://arxiv.org/abs/2509.01250.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025b. URL https://arxiv.org/abs/2506.05176.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group Sequence Policy Optimization. *arXiv preprint arXiv:2507.18071*, 2025. URL https://arxiv.org/abs/2507.18071.

Ziyu Zhu, Zhuofan Zhang, Xiaojian Ma, Xuesong Niu, Yixin Chen, Baoxiong Jia, Zhidong Deng, Siyuan Huang, and Qing Li. Unifying 3D Vision-Language Understanding via Promptable Queries. In *Computer Vision – ECCV 2024*, pp. 188–206. Springer, 2024. doi: 10.1007/978-3-031-72784-9_11. URL https://arxiv.org/abs/2405.11442.

APPENDIX OVERVIEW

This appendix gives the implementation details behind Sections 3 and 4, together with additional results. Appendices A–E follow the order of Section 3, from the model and the tools to the training data and the two training stages. Appendices F and G detail the evaluation protocols of Section 4, and Appendix H shows further qualitative results.

- **Appendix A: Model and Input Representation.** Defines how the input point cloud is sampled and normalized, and the code coordinates that tool feedback and programs share (Eq. (5)). It then specifies the Qwen3.5-2B backbone with the Point-PQAE encoder and point projector (Table 4), and the two-turn generation interface, including the tool-call format and the token budgets of both turns.
- **Appendix B: Segmentation and Measurement Tools.** Complements Section 3.3. It details the architecture, query encoding, and configuration of QDPS (Table 5) and its training loss (Eqs. (6) and (7)). For SGF, it gives the fixed thresholds (Table 6), the measurement routines for primitives and sketch extrusions, the response statuses and their serialization into the dialogue, and the rendering of the projection images.
- **Appendix C: Training Data Construction Details.** Complements Section 3.5. It covers the conversion of the source programs, the definition of step surface masks (Eq. (8)), the shape descriptions written by a vision-language model, the box perturbation used in tool replay, the length filtering that yields the SFT set, and the data splits, including the removal of DeepCAD test families from Omni-CAD. It closes with a complete training trajectory (Figure 5).
- **Appendix D: Training Configuration.** Describes the point-encoder adaptation and the projector alignment on Text2CAD that precede SFT, and lists the SFT configuration (Table 7).
- **Appendix E: Policy Optimization Objective and Configuration.** Complements Section 3.6. It gives the group-relative advantage, the GSPO-style sequence ratio and clipped loss, and the turn-wise KL regularizer (Eqs. (9)–(12)), followed by the solid-IoU reward with its execution checks (Eq. (13)) and the RL configuration (Table 8).
- **Appendix F: Evaluation Protocol.** Complements Section 4.1. It states the source of the CAD-SIGNet results and explains how CADReasoner is scored in Table 1, comparing its final-iteration and best-so-far selection rules (Table 9).
- **Appendix G: Tool Accuracy Details.** Complements the measurement-accuracy analysis in Section 4.3. It explains why the tools are evaluated with reference queries and which parameters count as observable, and reports per-step segmentation metrics (Table 10), SGF response statuses with predicted and ground-truth masks (Table 11), and measurement accuracy by parameter type (Table 12).
- **Appendix H: Additional Qualitative Results.** Shows further reconstructions on DeepCAD and Fusion360 (Figures 6 and 7), in the layout of Figure 4.

A MODEL AND INPUT REPRESENTATION

A.1 POINT SAMPLING AND COORDINATE TRANSFORMS

We sample 32,768 points on the final CAD surface and retain 8,192 using fixed farthest-point sampling (FPS). Let c be the input point cloud's bounding-box center and L its maximum axis length. For a point p, the normalized coordinates p_{norm} and the code coordinates p_{code} are

$$p_{\text{norm}} = \frac{2(p-c)}{L}, \qquad p_{\text{code}} = 100 p_{\text{norm}}, \qquad p = c + \frac{L}{200} p_{\text{code}}. \tag{5}$$

A single scale is shared by all three axes. The input points lie within $[-1, 1]^3$, and their longest axis spans 200 in code coordinates. Intermediate operation entities, cutting bodies, and program parameters can extend beyond this input range.

A.2 BACKBONE AND PROJECTOR

The policy uses Qwen3.5-2B (Qwen Team, 2026) with a Point-PQAE encoder (Zhang et al., 2025a). The point projector has the form

$$\mathrm{LN}(384) \to \mathrm{Linear}(384, 2048) \to \mathrm{GELU} \to \mathrm{Linear}(2048, 2048),$$

with approximately 4.99 million parameters. Its output supplies the policy's point tokens, and the native visual modules encode tool observation images.

Table 4: Point representation and multimodal backbone configuration.

Component	Configuration
Multimodal backbone	Qwen3.5-2B
Input point count	8,192
Local point groups	512
Points per group	32
Point encoder blocks	12
Point encoder width / attention heads	384 / 6
Point encoder output	512×384
Projected token dimension	2,048

A.3 GENERATION INTERFACE

The planning turn emits the complete plan and one `<tool_call>` object containing 1–10 queries. The code turn emits a complete CadQuery program after tool execution. The maximum numbers of newly generated tokens are 2,048 for the planning turn and 4,096 for the code turn. These per-turn generation limits are distinct from the full-trajectory SFT length limit.

A step is the complete operation entity and its `add`, `union`, `cut`, or `intersect` operation. A sketch extrusion normally constitutes one step. Queries and responses preserve this ordering and one-to-one correspondence. The step index orders the interaction and is also a query input of QDPS.

B SEGMENTATION AND MEASUREMENT TOOLS

B.1 QDPS ARCHITECTURE AND OBJECTIVE

Points are ordered along an XYZ Morton curve that does not depend on the order of the input array. Five local attention blocks process the ordered points in windows of 128 points; in each block, the query feature modulates the point features, and the blocks alternate unshifted windows and half-window shifts so that information crosses window boundaries. The query feature combines the bounding box, the shape category, the operation, the plane category (XY, YZ, or ZX), the step index, and the text embedding of the shape description. After the input point order is restored, a point-query compatibility head scores each point against the query.

Table 5: QDPS configuration.

Setting	Value
Network width / blocks / attention heads	320 / 5 / 5
Local window size	128 points
Text encoder	Qwen3-Embedding-0.6B (Zhang et al., 2025b)
Text input / projected dimension	1,024 / 256
Mask threshold η	0.25
Focal exponent	$\gamma = 2$
Focal / Dice / false-positive loss weights	$1/0.9/0.1$

For a query k with negative points (points with $m_{ki} = 0$ in Eq. (8)), let $\mathcal{N}_k$ be the indices of those points. For a predicted probability $\hat{m}$, define $\ell_{\mathrm{BCE}}(\hat{m}, 0) = -\log(1 - \hat{m})$. The false-positive term

is the mean over the negative points:

$$\mathcal{L}_{\mathrm{FP},k} = \frac{1}{|\mathcal{N}_k|} \sum_{i \in \mathcal{N}_k} \ell_{\mathrm{BCE}}(\hat{m}_{ki}, 0). \tag{6}$$

The query losses are averaged, and the full objective is

$$\mathcal{L}_{\mathrm{QDPS}} = \mathcal{L}_{\mathrm{focal}} + 0.9\mathcal{L}_{\mathrm{Dice}} + 0.1\mathcal{L}_{\mathrm{FP}}, \tag{7}$$

where $\mathcal{L}_{\mathrm{focal}}$ is the focal binary cross-entropy with exponent γ between $\hat{m}_{ki}$ and the target mask m_{ki} of Eq. (8), $\mathcal{L}_{\mathrm{Dice}}$ is the Dice loss between the predicted probabilities and the same mask, and $\mathcal{L}_{\mathrm{FP}}$ averages Eq. (6) over queries.

B.2 SGF MEASUREMENTS AND OBSERVATION IMAGES

SGF is a deterministic routine without learned parameters. Its input is the point set P_k selected by QDPS, and it returns values in the code coordinates of Eq. (5). It first transforms P_k into the frame of the query plane, whose in-plane axes u, v and normal n follow from the plane category, with n along the z, x, and y axes for XY, YZ, and ZX, respectively. The shape and operation in the query then select the measurement routine.

Implementation thresholds. The thresholds that determine whether a fit is accepted are fixed as follows. Here η is the QDPS mask-probability threshold, N is the number of selected points, r is a point-to-fit residual, s is the local scale used by the robust fitter, Δ_n is the extent along the query normal, R is the fitted circle or sphere radius, and r_c is the derived end-face separation cut-off. The residual resolution r_{res} is computed at runtime from the coordinate scale and floating-point precision; it is not an additional tuned constant. P90 denotes the 90th percentile of the normalized residuals, coverage denotes the fraction of relevant support bins or cells that pass the corresponding support test, and MAD denotes the median absolute deviation. The condition ratio, parameter uncertainty, and parameter stability are normalized diagnostics returned by the corresponding fit routines. Circle fitting uses random seed 0, and candidate generation and refinement use no other unrecorded randomness.

Table 6: Fixed thresholds and support checks in SGF.

Component	Fixed setting
Mask and input gate	QDPS threshold $\eta = 0.25$; at least $N \geq 8$ selected points.
Box extent	2% and 98% coordinate quantiles.
Robust fitting	Three iterations with $w = \exp[-(r/\sigma)^2]$; $\sigma = 0.035\ \mathrm{median}(s)$ for cylinders and $0.04\ \mathrm{median}(s)$ for boxes and sketch fits.
Planar support	An 8×8 grid with at least 6 points, 4 occupied cells, and coverage ≥ 0.20.
Exclusive cap support	At least 6 points, 4 exclusive cells, and coverage ≥ 0.10.
Sketch end-face clusters	At least 6 points per end; end-face distance $> 2r_c$, where $r_c = \max(2r_{\mathrm{res}}, 10^{-4}\Delta_n)$; interval point ratio ≥ 0.90.
Cylinder end support	At least 6 points per end; at least 50% of 12 angular sectors covered; at least 6 interior points per end.
Box face support	Surface inlier ratio ≥ 0.70, at least 4 points per face, and projected coverage ≥ 0.50; planar and cap support also meet the 6/4/0.20 and 6/4/0.10 gates above.
Circle fit	512 RANSAC candidates, at most 2 refinements, angular support $\geq 45^\circ$, condition ratio ≥ 0.08, and radial support ≥ 8.
Circle residual	Normalized residual P90 $\leq \max(0.03, 4r_{\mathrm{res}}/R)$; parameter uncertainty ≤ 0.03.
Sphere fit	At most 3 MAD refinements, condition ratio ≥ 0.10, parameter stability ≤ 0.03, and the same normalized residual P90 bound as the circle fit.
MAD trimming	$\mathrm{MAD}^* = 1.4826\ \mathrm{median}(\lvert r - \mathrm{median}(r)\rvert)$; retain $r \leq \mathrm{median}(r) + 4\max(\mathrm{MAD}^*, r_{\mathrm{res}})$.

Primitives. Box extents are the 2% and 98% coordinate quantiles of P_k, checked for surface support and residuals. A cylinder is measured by fitting a circle in the plane orthogonal to its axis, with checks on radial and angular support, support at both axial ends, and the fitting residual. A sphere is fitted linearly and checked by its inlier residuals. Each fit is refined robustly: inliers are updated with a threshold based on the median absolute deviation, so outliers reduce the support of a measurement instead of setting its extreme dimensions. Primitives that cut or intersect the part, such as holes, are measured from the surfaces they leave in the part.

Sketch extrusions. SGF measures the in-plane extents of P_k and then searches along the normal for two end-face clusters. Each cluster needs enough points, planar area, and in-plane coverage, and an exclusive support test separates end faces from adjacent surfaces. The two faces must also be sufficiently far apart and supported in between. The start, end, and depth of the extrusion are returned only when both end faces pass these tests.

Responses. Every routine requires at least 8 points. A response is a success when its measurements are complete and supported, partial when only the directly observable quantities are returned, and a failure when it returns no measurement but an error or warning. Unsupported fields are omitted, and each response records the number of selected points and its support diagnostics. The responses to all queries are serialized as structured text in one tool-result message, which is appended to the dialogue together with the projection images of the sketch-extrude steps; the code turn follows this message. At inference and during RL, the tools answer the queries that the policy emits. Reference queries are used only to build the training trajectories (Appendix C.4) and to evaluate the tools (Appendix G).

Observation images. A sketch observation is an orthographic projection of the point cloud onto the query's (u, v) plane, rendered at 336×336 pixels with equal-scale axes and numeric ticks. The points selected by QDPS are highlighted, and the remaining points are shown in gray as context.

C Training Data Construction Details

C.1 Source programs

CAD-Recode (Rukhovich et al., 2025) programs are parsed and regenerated in a canonical CadQuery style. Omni-CAD samples from CAD-MLLM (Xu et al., 2024a) are stored as DeepCAD-style JSON command and parameter sequences, which a deterministic translator converts into the same style. Executing the unified programs step by step yields the final surface, the operation entity of every step, and the geometric fields of the modeling plan.

C.2 Step surface masks

Let O_k be the complete operation entity of step k, and let ∂O_k be its surface. For a normalized final-surface point p_i, with $d(\cdot, \cdot)$ the unsigned point-to-surface distance and $\mathbf{1}[\cdot]$ the indicator function, the target is

$$m_{ki} = \mathbf{1}[d(p_i, \partial O_k) \le \tau_k], \qquad \tau_k = \max\big(0.008,\, 0.006\|b_k^{\max} - b_k^{\min}\|_2\big). \tag{8}$$

Here, $b_k^{\min}$ and $b_k^{\max}$ bound the complete operation entity in normalized coordinates. The distance is computed against its triangulated surface. The same surface-distance rule applies to cutting and intersecting entities. Sampling from the final solid determines which surface points are available for supervision. Shared surfaces and points within the geometric tolerance can have positive labels for multiple steps.

C.3 Shape descriptions

A vision-language model, GPT-5.4 mini (OpenAI, 2026), describes sketch steps. Its main inputs are images of the step, with the step's CadQuery fragment as auxiliary structural context. The annotation prompt prioritizes the images and requests 5 to 29 English words without dimensions, coordinates, API names, or operation words. Recognizable simple primitives use rules or existing primitive metadata. The construction pipeline checks each description against its step entity.

C.4 Box perturbation and tool replay

Query boxes receive a random 5% size perturbation before replay, so that QDPS and the policy are trained on the imprecise boxes that a planning turn produces at inference time. The perturbation applies only to the queries; the target programs keep their dimensions. The perturbed box is clipped to the code-coordinate range $[-100, 100]$ of Eq. (5).

QDPS and SGF then produce the text and image observations. Partial or failure status alone does not remove a trajectory: a complete call–feedback–code chain can be retained after the source, geometry, and length checks. Execution-environment faults and missing feedback are excluded.

C.5 Trajectory pool and sequence filtering

Trajectory construction covers CAD models with at most 10 steps and yields 1,263,407 trajectories. The SFT limit is 4,096 tokens for the complete multimodal sequence, including plan, calls, feedback, code, and actual point/image token costs. Overlength trajectories are filtered as complete units. The remaining 1,189,612 trajectories form the SFT set: 809,646 from CAD-Recode and 379,966 from CAD-MLLM, approximately 68.06% and 31.94%, respectively.

C.6 Data splits

The SFT trajectories are built from CAD-Recode (Rukhovich et al., 2025) and from Omni-CAD, the dataset of CAD-MLLM (Xu et al., 2024a). CAD-Recode programs are procedurally generated. Omni-CAD takes its designs from the same source as DeepCAD and parses them with DeepCAD's pipeline, but keeps designs with chamfers or fillets by removing those operations, so it contains the DeepCAD designs. It also expands each design of its training split into a family of models, the intermediate designs after each of its extrusions. Before training, we remove every Omni-CAD family whose design is in the DeepCAD test set. The Text2CAD samples used for projector alignment (Appendix D) also exclude the DeepCAD test models. RL uses only models from the official training splits of DeepCAD and Fusion360 (Appendix E). All results are reported on the official test sets, which contain 8,046 models for DeepCAD and 1,725 for Fusion360.

C.7 Example trajectory

Figure 5 shows a trajectory from the training set.

Example training trajectory

P Input and prompt

Point cloud with 8,192 points, encoded as 512 point tokens.
User: "Reconstruct the CAD model from the point cloud. First write a concise construction plan [...] After the tool_result observation, write executable CadQuery code inside code tags."

A Plan and calls

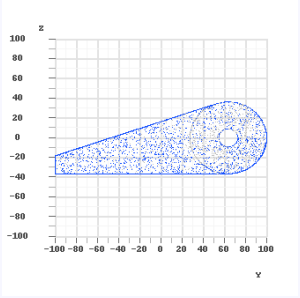
The plan lists the operation, shape, plane, axis, box, and description of each step, and one tool call sends them as queries:

```
{"calls": [
 {"tool": "observe_sketch_extrude", "step": 1, "op": "add",
  "shape": "sketch_extrude", "plane": "YZ", "axis": "X",
  "bbox": [-38.899, -99.167, -37.656, 36.527, 100, 33.34],
  "desc": "A wedge-like block with a rounded end and one off-center
           circular hole near that end"},
 {"tool": "measure_hole_or_cut", "step": 2, "op": "cut",
  "shape": "cylinder", "plane": "YZ", "axis": "X",
  "bbox": [-17.193, 28.309, -37.238, 18.38, 100, 35.155],
  "desc": "X-axis cylindrical hole"}
]}
```

F Tool feedback

Step 1 (`observe_sketch_extrude`): success
plane YZ, normal axis X, depth 72.705, range $[-36.353,\ 36.353]$, projection image (right)

Step 2 (`measure_hole_or_cut`): success
axis X, radius 36.329, height 37.549, center $(0.073,\ 63.628,\ -0.017)$, kind hole



C CadQuery target

```
import cadquery as cq
wp0 = cq.Workplane('YZ', origin=(0, 0, 0))
step_000 = (wp0.sketch()
    .segment((-100, -36.364), (63.636, -36.364))
    .arc((99.496, 6.033), (51.738, 34.362))
    .segment((-100, -18.182)).segment((-100, -36.364)).assemble()
    .reset().push([(63.636, 0)]).circle(9.091, mode='s')
    .finalize().extrude(36.364, both=True))
step_001 = wp0.workplane(offset=0).moveTo(63.636, 0).cylinder(36.364, 36.364)
result = step_000.cut(step_001)
```

Figure 5: **A training trajectory.** The four parts correspond to (P, A, F, C) in Section 3.5. The projection image in F is returned for the sketch-extrude step. Lengths are in code units.

D Training Configuration

D.1 Point encoder and projector

The point encoder is initialized from Point-PQAE weights and adapted to CAD surfaces by local point reconstruction. The projector is then aligned with the language model on Text2CAD (Khan et al., 2024b) samples, with the point encoder, language model, and visual modules frozen. SFT starts from these weights and makes all components trainable.

D.2 Supervised fine-tuning

SFT runs for three epochs on the 1,189,612 trajectories of Appendix C (Table 7).

Table 7: SFT configuration.

Setting	Value
Epochs / updates	3 / 55,764
GPUs / per-device batch / accumulation	$8/1/8$
Language-model learning rate	10^{-5}
LR schedule / warmup	Cosine / 2,789 updates
Precision / distributed optimizer	BF16 / ZeRO-1
Maximum complete sequence length	4,096

E Policy Optimization Objective and Configuration

The RL stage follows SFT. The actor is initialized from the SFT policy, with a frozen copy as the KL reference. QDPS and SGF remain fixed throughout the rollouts.

E.1 Group-relative sequence objective

For an input, let $\mathcal{V}$ be the set of trainable candidates among the G sampled candidates, and let R_i be the reward of candidate i from Eq. (4). Policy-invalid candidates remain in this set with reward -1; infrastructure failures are excluded. The advantage is mean-centered without division by the group standard deviation:

$$\hat{A}_i = R_i - \frac{1}{|\mathcal{V}|} \sum_{j \in \mathcal{V}} R_j. \tag{9}$$

Let $\mathcal{Y}_i$ contain the valid policy tokens from the planning and code turns together. The GSPO-style importance ratio (Zheng et al., 2025) is

$$\rho_i(\theta) = \exp\left[\frac{1}{|\mathcal{Y}_i|} \sum_{t \in \mathcal{Y}_i} \left(\log \pi_\theta(y_{it} \mid h_{it}) - \log \pi_{\text{old}}(y_{it} \mid h_{it}) \right) \right]. \tag{10}$$

Here, y_{it} is the t-th generated token of candidate i, h_{it} is its prefix together with the available point, image, and tool context, and π_{old} is the policy that sampled the candidates. Both turns share the terminal reward and advantage. Their generated tokens contribute to one length-normalized ratio; tool observations and input tokens are excluded from $\mathcal{Y}_i$.

The clipped policy loss is

$$\mathcal{L}_{\text{policy}} = -\frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \min\left[\rho_i \hat{A}_i,\ \text{clip}(\rho_i, 1-\epsilon_l, 1+\epsilon_h) \hat{A}_i \right], \tag{11}$$

where $\hat{A}_i$ and ρ_i are given by Eqs. (9) and (10), and ϵ_l and ϵ_h are the lower and upper clipping widths. Let $\mathcal{Y}_i^A$ and $\mathcal{Y}_i^C$ denote the generated tokens in the planning and code turns, and κ_{it} the token-level KL term relative to the frozen SFT reference. The trajectory regularizer is

$$\Omega_i = \frac{1}{|\mathcal{Y}_i^A|} \sum_{t \in \mathcal{Y}_i^A} \kappa_{it} + \lambda_C \frac{1}{|\mathcal{Y}_i^C|} \sum_{t \in \mathcal{Y}_i^C} \kappa_{it}, \qquad \lambda_C = 1, \tag{12}$$

where λ_C weights the code-turn term. The overall objective adds $\beta|\mathcal{V}|^{-1}\sum_{i\in\mathcal{V}}\Omega_i$ (Eq. (12)) to $\mathcal{L}_{\text{policy}}$ in Eq. (11), where β is the KL coefficient.

E.2 SOLID-IOU REWARD AND EXECUTION CHECKS

The geometric reward uses triangulated B-Rep solids and Manifold mesh Boolean intersection:

$$\text{IoU}_{\text{solid}} = \frac{V_{\cap}}{V_{\text{gt}} + V_{\text{pred}} - V_{\cap}}, \tag{13}$$

where V_{gt} and V_{pred} are the volumes of the target solid $S^{\star}$ and the reconstruction S_i, and $V_{\cap}$ is the volume of their intersection. Triangulation uses linear deflection 0.001 and angular deflection 0.1. This is a mesh approximation of the intersection volume. Candidate checks require valid code and execution, a valid B-Rep, positive finite volume, finite spatial bounds, and a usable closed mesh. The mesh-to-B-Rep relative volume error threshold is 1%, and the self-IoU error threshold is 10^{-5}.

Multiple solids are allowed. Triangulation preserves B-Rep solid component boundaries so that cavity shells are treated as part of their owning solids. Chamfer distance is recorded as a diagnostic and does not enter the reward. There are no separate dimension-error, plan-correctness, or tool-success reward terms. This training reward is distinct from the final benchmark evaluation protocol.

E.3 RL CONFIGURATION

RL uses 10,000 training models: 5,000 from DeepCAD and 5,000 from Fusion360. The policy, point encoder, projector, and visual modules are updated jointly with the configuration in Table 8.

Table 8: RL configuration.

Setting	Value
RL data	DeepCAD 5,000 + Fusion360 5,000 models
Total updates	1,000
Candidates per input G	16
Per-device batch / gradient accumulation	$2/10$
Main / projector learning rate	7×10^{-7}
Point encoder / visual learning rate	7×10^{-8}
LR schedule / minimum ratio / warmup	Cosine / 0.2 / 20 updates
Clip widths (ϵ_l, ϵ_h)	$(0.0003,\ 0.0004)$
Initial KL coefficient β_0	0.01
KL controller	Range $[0.005, 0.05]$; target 0.015; EMA decay 0.95
Planning / code token budget	2,048 / 4,096
Sampling	Temperature 1.0; top-p 0.95
Tool-call budget	Maximum 10 queries per trajectory
Reward	Solid IoU (Eq. (13)); -1 for invalid candidates
Precision / attention / optimizer	BF16 / FlashAttention-2 / ZeRO-2
Gradient clip	1.0
AdamW (β_1, β_2)	$(0.9,\ 0.95)$
Seed	42

F EVALUATION PROTOCOL

Unless a benchmark protocol explicitly states otherwise, all final evaluations use greedy decoding at inference time. The temperature and top-p settings in Table 8 apply to RL training rollouts, not to benchmark evaluation.

The CAD-SIGNet (Khan et al., 2024a) results in Table 1 are taken from the cadrille paper (Kolodiazhnyi et al., 2025), which reports them without test-time sampling and with fewer decimal places; we append zeros to match the precision of the table. CAD-Recode is evaluated with its publicly released v1.5 model.

F.1 SCORING OF CADREASONER

CADReasoner (Kabisov et al., 2026a) edits its program over t iterations. Its evaluation protocol reports, at every step, the best-so-far program $C_{\leq t} = \arg\min_{i \leq t} D(T_{\text{eval}}, \text{Exec}(C_i))$, where D is the Chamfer distance between the target surface T_{eval} and the executed candidate $\text{Exec}(C_i)$; on the clean track, T_{eval} is the target mesh and, in the authors' words, "both selection and reporting use the clean target". The reported $t = 5$ numbers therefore select one of five programs with the evaluation metric, and a shape counts as invalid only if all five iterations fail. The other methods in Table 1 produce one program per shape.

For the single-output comparison in Table 1, we run the released CADReasoner point-cloud model with greedy decoding for $t = 5$ iterations, where each iteration edits the program of the previous iteration, and score the program of the final iteration. Test sets and metric computation are identical to those used for the other methods. We also evaluate the same five iterations with the original best-so-far rule, which uses the clean target to select one of the five programs. The two selection rules give the following results:

Table 9: CADReasoner under two five-iteration selection rules. The final-iteration rows are the values used in Table 1; the best-so-far rows are our rerun with target-based selection.

Dataset	Selection	CD↓	IoU↑	IR↓
DeepCAD	Final iteration	0.194	89.0	1.72
	Best-so-far (target)	0.170	92.1	0.012
Fusion360	Final iteration	0.216	81.7	2.67
	Best-so-far (target)	0.179	86.7	0.000

Best-so-far selection can substantially lower IR because a shape is counted as valid when any of its five candidates succeeds; this is a protocol effect rather than a single-output validity comparison. Under the best-so-far rule, QM-CAD remains better on CD and IoU on both benchmarks, while CADReasoner has the lower IR by this selection rule. For reference, the original paper reports, for its point-cloud model on the clean track (greedy decoding; its Table 2), median CD / IoU / IR of 0.21 / 86.1 / 1.5 at $t = 1$ and 0.17 / 89.2 / 0.2 with best-so-far selection at $t = 5$ on DeepCAD, and 0.22 / 77.3 / 3.2 and 0.16 / 83.1 / 0.7 on Fusion360.

G TOOL ACCURACY DETAILS

This section complements Table 3. The evaluation covers the full DeepCAD test set, with each step issued with its reference query, i.e., the query of the corresponding step in the reference program. A parameter counts as observable when SGF returns that field from the ground-truth step mask. Thus, the evaluation asks how many parameters recoverable by SGF under the reference mask are also returned by SGF and measured accurately when it uses the predicted mask. Coverage and tolerance accuracy use all reference-mask-observable parameters as the denominator, and errors are computed over the subset of these parameters that SGF returns from the predicted mask. Errors are absolute errors in code units (Eq. (5)), in which the longest axis of the input spans 200 units.

We use reference queries because CAD decompositions are not unique. A hollow cylinder, for example, can be built as one extruded annulus or as a cylinder minus a coaxial cylinder; both programs produce the same solid, but their steps, step surface masks, and parameters differ. The masks of Eq. (8) are defined for one decomposition, so a query generated by the policy may have no counterpart in the reference program. Reference queries fix the measurement target and evaluate the tools in isolation, while the reconstruction metrics in Section 4.2 evaluate the full system with the queries that the policy generates.

Segmentation. QDPS returns an empty mask for 0.15% of the steps, and 99.33% of its masks contain at least 8 points, the minimum that SGF requires. Table 10 reports the mean and median of the per-step mask metrics.

Measurement status. Table 11 compares the SGF status obtained from predicted masks with that obtained from ground-truth masks. The two aggregate distributions are close, with complete re-

Table 10: Per-step segmentation metrics of QDPS (%).

Metric	Mean	Median
IoU	94.6	99.2
Precision	96.2	99.9
Recall	98.1	99.8
F1	96.7	99.6

sponses in 41.6% of steps for both masks. Even with ground-truth masks, 57.5% of steps remain partial, indicating that improving segmentation alone does not guarantee a complete SGF measurement under its support conditions.

Table 11: SGF response status with predicted and ground-truth step masks (% of steps).

Status	Predicted masks	Ground-truth masks
Success	41.6	41.6
Partial	57.7	57.5
Failure	0.8	0.9

Parameters. Table 12 reports coverage, tolerance accuracy, and errors for each parameter type; Table 3 pools these results.

Table 12: Measurement accuracy by parameter type. Cov. is the fraction of parameters returned by SGF with the ground-truth step mask that are also returned with the predicted mask. Extrusion start and end are the positions of the two end faces along the plane normal.

		Accuracy (%)↑			Error (code units)↓	
Parameter	Cov. (%)↑	@0.1%L	@0.5%L	@1%L	Mean	Median
Box center	97.2	89.4	96.9	97.1	0.0457	0.0005
Box size	97.2	80.4	84.9	87.8	0.4393	0.0215
Cylinder center	98.6	95.3	98.0	98.4	0.0544	0.0082
Cylinder height	98.6	96.9	97.9	98.4	0.0232	0.0018
Cylinder radius	99.0	85.3	97.9	98.5	0.1218	0.0206
Extrusion depth	95.1	92.5	94.8	94.9	0.1057	0.0037
Extrusion start	95.1	93.3	94.9	95.0	0.0712	0.0019
Extrusion end	95.1	94.3	95.0	95.0	0.0354	0.0017

H Additional Qualitative Results

Figures 6 and 7 show further reconstructions on DeepCAD and Fusion360, in the layout of Figure 4. The baselines often miss or distort repeated holes, as on the perforated bars and plates, split a part into pieces, or bend its outline, as when a round disc becomes a hexagon. QM-CAD recovers the hole patterns and outlines of these parts and keeps each part in one piece.

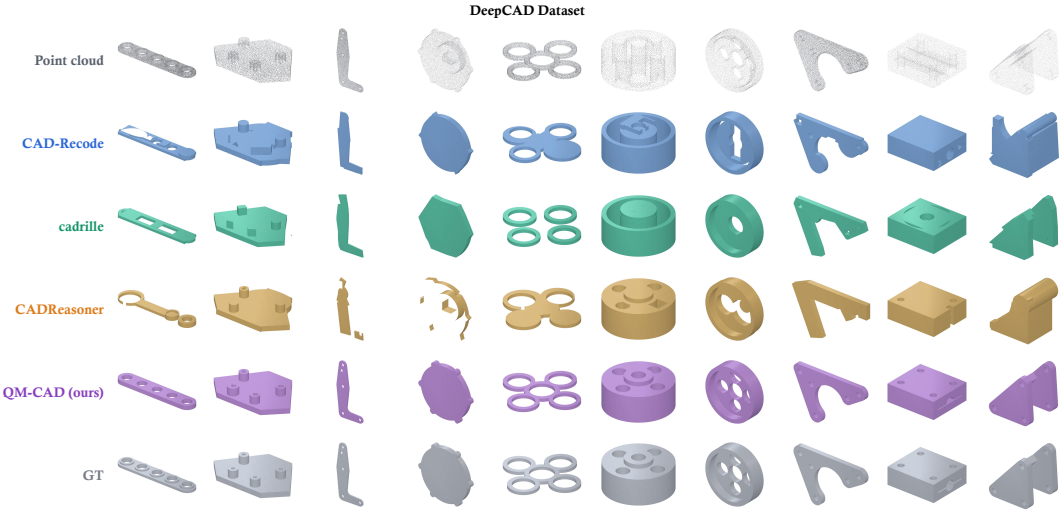


Figure 6: **Additional qualitative results on DeepCAD.** Parts from DeepCAD (Wu et al., 2021). Rows from top to bottom: input point cloud, CAD-Recode (Rukhovich et al., 2025), cadrille (Kolodiazhnyi et al., 2025), CADReasoner (Kabisov et al., 2026a), QM-CAD, and the ground truth.

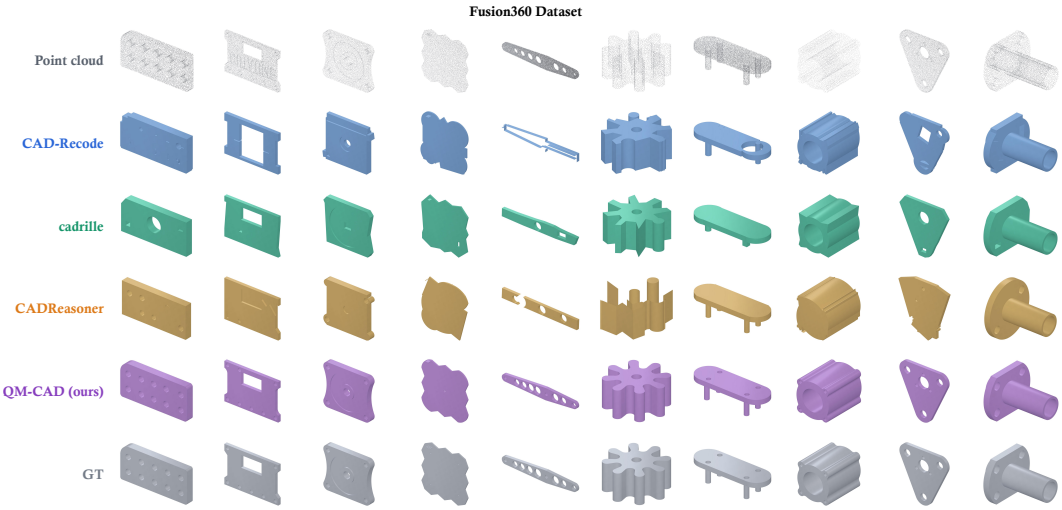


Figure 7: **Additional qualitative results on Fusion360.** Parts from Fusion360 (Willis et al., 2021). Rows from top to bottom: input point cloud, CAD-Recode (Rukhovich et al., 2025), cadrille (Kolodiazhnyi et al., 2025), CADReasoner (Kabisov et al., 2026a), QM-CAD, and the ground truth.