CADRE: AGENTIC CAD REVERSE ENGINEERING WITHOUT 3D ENCODERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reverse engineering 3D Computer-Aided Design (CAD) shapes is a fundamental challenge in modern engineering workflows, which aims to recover editable construction sequences from static geometry. Large language models are a natural fit for this task because they can synthesize procedural CAD programs, which can be represented as Python code. However, they cannot natively support 3D CAD as a modality, since there are no pretraining data comparable in scale and diversity to image and text data. Prior work bridges this modality gap by training 3D encoders to align geometry with the language model. Yet the alignment is itself a difficult task due to data scarcity, and these approaches tie the encoder to a fine-tuned model, precluding the use of frozen frontier LLMs. We observe that CAD geometry need not be learned as a new modality. An LLM can write programs to read or estimate its structures and dimensions, and render it to grasp the overall shape, so the input arrives entirely as text and images. Building on this observation, we propose **CADre**, a multi-agent framework for CAD reverse engineering that equips frontier LLMs with a Python environment and a renderer, allowing them to look at and query CAD shapes directly without training a 3D encoder. In our system, a cadre of specialized agents investigate the target geometry, generate CAD programs, and verify the resulting reconstructions collaboratively. Experiments on three CAD datasets show that our system substantially outperforms prior state-of-the-art methods by up to 38 IoU points on the hardest shapes.

1 INTRODUCTION

Computer-Aided Design (CAD) is the central component in engineering and manufacturing, allowing engineers to create precise 3D models for production. However, editing a CAD model usually requires modifying the construction sequence that produced the shape, rather than directly editing the shape itself at the geometric level. This is analogous to software engineers editing source code rather than a compiled binary. In practice, this construction history is often lost when models are exchanged as STEP files, which preserve geometry in a form of Boundary Representations (B-rep) (Weiler, 1986), but discard the original modeling operations. CAD reverse engineering addresses this gap by recovering editable code from static geometry.

Traditionally, CAD reverse engineering is a time-consuming process that requires engineers to iteratively inspect the shape, modify the code, and verify the result. This has motivated recent work on using LLMs for CAD reverse engineering since the problem can be cast as code generation. However, standard models do not natively accept 3D CAD geometry as input. Previous work addresses this limitation by training a 3D encoder to map the geometry to a representation that LLMs can consume. For example, CAD-Recode (Rukhovich et al., 2025) fine-tunes a small LLM with a point cloud encoder to generate CAD code in a single inference pass. Cadrille (Kolodiazhnyi et al., 2025) extends this with multimodal inputs and reinforcement learning fine-tuning, but remains a single-pass system. CAD-Reasoner (Kabisov et al., 2026) trained a model for iterative code refinement conditioned on multi-view depth renders of the shape and point clouds.

Despite their differences, these methods rest on the same foundation, a 3D encoder trained to align geometry with the language model. This alignment is itself a hard problem. Even with billions of image-text pairs, vision-language models still fail at perceiving simple geometric details and hallucinate content absent from the input (Tong et al., 2024; Rahmanzadehgervi et al., 2025; Li

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.

et al., 2023b). For 3D CAD, where paired data are orders of magnitude scarcer (Xu et al., 2024), the challenge is only harsher. Moreover, an aligned encoder is tied to the model with which it was tuned, precluding the use of frozen frontier LLMs.

In this work, we ask:

Can frontier LLMs directly understand CAD geometry without a 3D encoder?

Our key observation is that CAD geometry does not need to be learned as a new modality since it is already a structured, queryable object. For instance, a B-rep file stores analytic geometry, so we can directly read structural quantities such as the number of faces or the radius of a cylindrical face by writing programs. Even when we are given a point cloud, a plain N-by-3 array, we can use computer programs to estimate its structure (see Figure 2). Rendering can also complement these queries with a global view to reveal the overall form and semantics of the shape. All of this information can be consumed by the LLM as text and images. No new modality is needed.

We therefore replace the 3D encoder with a Python environment and a renderer, reformulating 3D perception as an active querying process rather than a learned encoding. Concretely, we build an agentic system with three roles. An *investigator* agent inspects the target shape over multiple turns and summarizes its findings into a structured text description and a plan for reconstruction. A *generator* agent takes this description and writes programs in CadQuery (contributors, 2026), a Python library for procedural CAD modeling, to reconstruct the shape. A *verifier* agent then checks the reconstructed shape against the target through the same tools and returns feedback that guides the next revision. Together, the three agents automate the inspect, modify, and verify loop that human engineers traditionally perform by hand. Since nothing in our pipeline is fine-tuned, we can use frozen frontier LLMs directly. Meanwhile, the multi-turn refinement loop steadily improves reconstruction quality, as shown in our experiments.

Our system substantially outperforms prior methods on all three datasets (DeepCAD, Fusion 360 and ABC) on 3D IoU and Chamfer Distance. Unlike previous approaches that require task-specific fine-tuning and custom model architecture, our framework leverages an off-the-shelf frontier LLM and works out of the box. Moreover, since the method inherits the reasoning and code-generation capabilities of whatever base model it uses, its performance improves automatically as frontier models advance. Notably, the same agentic loop remains effective even when the input is given as a point cloud rather than B-rep, where exact geometry can no longer be queried directly and must be estimated from sampled points.

In summary, our contributions are as follows:

- We bypass the modality alignment problem in CAD reverse engineering by treating CAD geometry as a queryable object rather than a new modality, allowing an LLM to inspect it directly through program execution and rendering.
- Building on this idea, we propose CADre, a multi-agent framework for CAD reverse engineering in which a cadre of specialized agents investigate the target geometry, generate CAD programs, and verify the resulting reconstructions. Because it is training-free, our method runs on frozen frontier LLMs without any fine-tuning or custom 3D architecture.
- We conducted extensive experiments on three standard CAD benchmarks under both B-rep and point-cloud inputs, showing that our approach substantially outperforms prior fine-tuned methods.

2 Related Work

CAD Reverse Engineering. CAD reverse engineering recovers a parametric construction sequence from a 3D shape. Early methods represent models as Constructive Solid Geometry (CSG) trees composed of Boolean operations over geometric primitives (Sharma et al., 2018; Du et al., 2018; Kania et al., 2020; Yu et al., 2024). While CSG enables reconstruction, it diverges from how engineers actually build CAD models in practice. More recent approaches adopt the sketch-extrude representation, which better aligns with modern CAD workflows. Wu et al. (2021) introduced a transformer-based autoregressive model for generating sketch-extrude sequences from point clouds

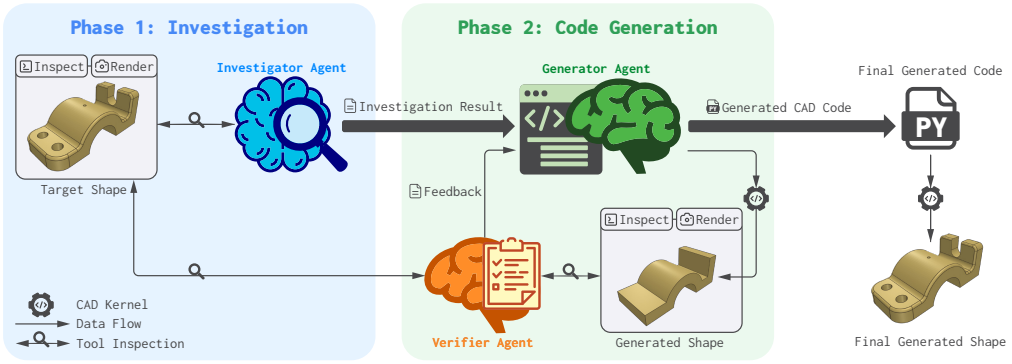


Figure 1: Overview of our agentic CAD reverse engineering system. In the first phase, the investigator inspects the target geometry over multiple turns through Python queries and rendering, then summarizes its findings as a structured description and reconstruction plan. In the second phase, the generator synthesizes CAD code over multiple refinement turns using this description and feedback from the verifier where the verifier performs similar inspection on both the target and predicted shapes to identify discrepancies.

and released a large-scale dataset of 180k CAD models. Subsequent works explored various architectural improvements for sketch-extrude prediction from point clouds or voxels (Lambourne et al., 2022; Uy et al., 2022; Ren et al., 2022; Li et al., 2023a; Ma et al., 2024; Khan et al., 2024a; Dupont et al., 2024). A separate line of work takes B-rep directly as input (Zhang et al., 2024; Kim et al., 2026). All of these methods share a common pattern: a dedicated 3D encoder (for point clouds or B-rep) is jointly fine-tuned with a decoder to produce the construction sequence in a single forward pass over the encoded geometry.

LLM-Based CAD Reverse Engineering. A recent line of work applies LLMs to CAD reverse engineering by representing sketch-extrude sequences as executable Python code rather than custom token formats. Rukhovich et al. (2025) fine-tunes a lightweight LLM (Qwen2-1.5B) with a point cloud projector to generate CadQuery (contributors, 2026) code from point clouds. Kolodiazhnyi et al. (2025) extends this setting to point clouds, images, and text within a single model, and applies reinforcement learning fine-tuning with geometric feedback to improve reconstruction quality. Kabisov et al. (2026) fine-tunes a VLM via a curriculum of on-policy rollouts to iteratively rewrite CadQuery code conditioned on multi-view depth renderings. These methods constitute the most direct baselines for our task. A broader body of work generates CAD code from images (Chen et al., 2025; You et al., 2025; Elistratov et al., 2026) or text descriptions (Khan et al., 2024b; Wang et al., 2025a; Guan et al., 2025; Xie & Ju, 2025; Govindarajan et al., 2026), or unifies multiple input modalities within a single model (Xu et al., 2025; Wang et al., 2025b; Li et al., 2025). However, these address prompt-conditioned generation rather than recovering the construction sequence of a given 3D geometry, and are therefore not directly comparable to our setting. Although the reverse engineering methods above successfully leverage pre-trained LLMs and code-based representations, they all require end-to-end fine-tuning, which prevents the use of frozen frontier LLMs whose weights cannot be modified.

Agentic CAD Systems. Recent works have begun exploring LLM agents for CAD tasks. Alrashedy et al. (2025) iteratively verifies and refines VLM-generated CAD code by rendering the generated object and prompting the VLM with visual validation questions, Panta et al. (2025) employs a multi-agent system with specialized roles for text-to-CAD generation, and Li et al. (2026) uses DeepSeek-R1 with visual and chain-of-thought feedback for self-refined CAD generation. In all three systems, the target shape is specified only through text, with no channel for ingesting a target 3D shape, so these frameworks do not extend to reverse engineering where the ground-truth geometry itself must be perceived and matched. Closer to our setting, Mallis et al. (2024) augments a VLM with CAD-specific tools that can operate on given CAD files, but targets interactive assistance across diverse tasks such as question answering and sketch parameterization, with 3D reconstruction limited to demos rather than a systematic treatment. Our work builds on the codebase of Yin et al. (2026), a dual-agent framework that reconstructs input images as editable Blender (Blender Online Community, 2026) scene programs through an iterative code-render-inspect loop, and we adopt their

sliding-window memory scheme for our own agents. However, their agent is conditioned on images and perceives geometry purely through renderings of its own generated scene. In contrast, we target CAD reverse engineering, where the input is the 3D geometry itself, and equip our agents with programmatic inspection of the target shape in addition to rendering. Our system is the first to combine a multi-agent framework with a Python environment that lets frozen frontier LLMs actively query 3D geometry and perform iterative CAD reverse engineering without any fine-tuning or custom 3D encoder.

3 METHOD

3.1 PROBLEM FORMULATION

CAD reverse engineering may start from different forms of 3D data. In some engineering workflows, the exact B-rep is available from a STEP file, while in others only scanned points or sampled geometry are accessible. We therefore consider two input settings. We denote the target CAD geometry by $\mathcal{B}$. The available input $\mathcal{X}$ for our system is either the B-rep itself, $\mathcal{X} = \mathcal{B}$, or a point cloud sampled from the surface of the shape, $\mathcal{X} = \mathcal{P} = \texttt{Sample}(\mathcal{B}) \in \mathbb{R}^{N\times 3}$. The goal is to recover Python code c such that

$$\hat{\mathcal{B}} = \texttt{Exec}(c) \approx \mathcal{B}, \tag{1}$$

where `Exec` denotes the CadQuery kernel that compiles c into a predicted shape $\hat{\mathcal{B}}$.

3.2 SYSTEM OVERVIEW

As shown in Figure 1, our system adapts the generate–verify agent loop of Yin et al. (2026) to CAD reverse engineering with a dedicated investigation stage and a CAD-specific tool environment (Section 3.4). Three agents communicate through text. The investigator first analyzes the target B-rep $\mathcal{B}$ by writing and executing Python code against the CadQuery API to measure exact geometric properties (e.g., radii and axial extents of cylindrical faces, as in Figure 2), and by rendering the shape from chosen viewpoints. It then summarizes its findings into a structured text description

$$d = \textsc{Investigator}(\mathcal{B}), \tag{2}$$

which persists through the rest of the run. The generator and verifier then alternate for up to T rounds: at each round $t = 1, \ldots, T$,

$$\begin{aligned} c^{(t)} &= \textsc{Generator}(d, c^{(<t)}, f^{(<t)}), \\ \hat{\mathcal{B}}^{(t)} &= \texttt{Exec}(c^{(t)}), \\ f^{(t)} &= \textsc{Verifier}(\mathcal{B}, \hat{\mathcal{B}}^{(t)}, c^{(t)}), \end{aligned} \tag{3}$$

where $c^{(t)}$ is CadQuery code, $\hat{\mathcal{B}}^{(t)}$ is the resulting predicted shape, and $f^{(t)}$ is text feedback. We write $c^{(<t)} = \{c^{(s)}\}_{s<t}$ and $f^{(<t)} = \{f^{(s)}\}_{s<t}$ for the history of prior codes and feedback, with $c^{(<1)} = f^{(<1)} = \emptyset$. The loop terminates when the verifier thinks $\hat{\mathcal{B}}^{(t)} \approx \mathcal{B}$ or when the budget T is exhausted. To bound context length, we follow the sliding-window memory scheme of Yin et al. (2026) where the generator is conditioned on the most recent K messages of its history, and the verifier is reset between rounds and conditioned only on the inputs shown in Eq. 3.

3.3 AGENT ROLES

Investigator. The investigator implements INVESTIGATOR in Eq. 2, performing the initial 3D encoding that converts the target shape into a text description usable by any frontier LLM. Operationally, it is a multi-turn agent that executes custom Python inspection code to probe specific details, renders the input shape from arbitrary viewpoints, and queries its geometric statistics. It then organizes its findings into d, which has two parts: an *overall description* covering appearance, geometric details, and notable features (e.g., holes, fillets, chamfers), and a *detailed reconstruction plan* to guide later reconstruction steps. Figure 2 illustrates this process on a representative example, contrasting how the agent adapts its inspection strategy to the input modality: directly querying face geometry for B-rep, and recovering the same quantities statistically from point clouds.

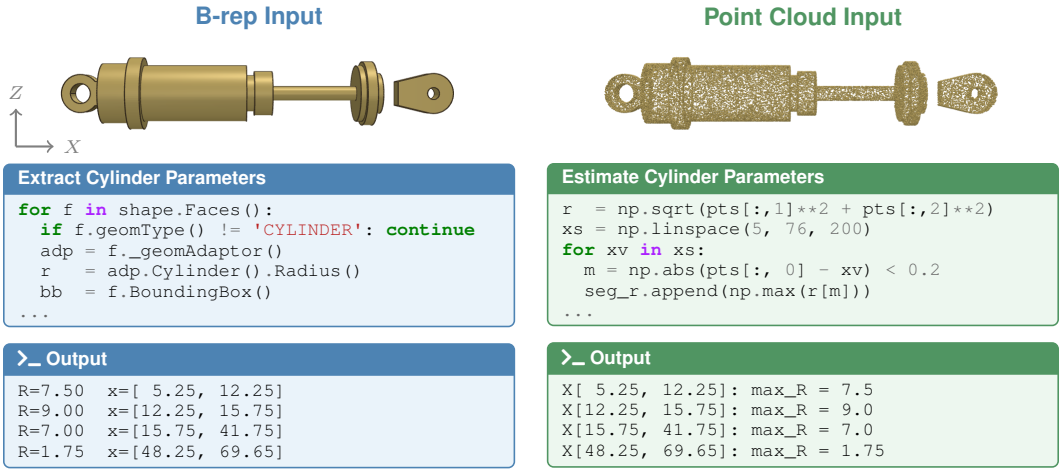


Figure 2: The investigator agent inspects 3D geometry through tool use rather than a learned encoder. With **B-rep** input, it extracts exact cylinder parameters from analytic faces; with **point cloud** input, it estimates the same parameters by sweeping along the axis and measuring the radial profile. These measurements are converted into a structured shape description and reconstruction plan for the generator.

Generator. The generator implements GENERATOR in Eq. 3 as a single LLM call with no images or tool access. Its task is pure code synthesis from a text specification. If $\texttt{Exec}(c^{(t)})$ raises a runtime error, the traceback is appended to the context and the generator is re-invoked within the same round to fix the failure before proceeding to verification.

Verifier. The verifier implements VERIFIER in Eq. 3 as a multi-turn agent: it inspects $\mathcal{B}$ and $\hat{\mathcal{B}}^{(t)}$ using the same tool suite as the investigator, now applied to both shapes, and is also shown the current code $c^{(t)}$ so it can reason about which lines to modify. It runs custom comparison code, renders $\mathcal{B}$ and $\hat{\mathcal{B}}^{(t)}$ side by side from matching viewpoints, and queries the statistics of both to detect numerical discrepancies (e.g., mismatched face counts or volume differences). The resulting feedback $f^{(t)}$ identifies specific geometric discrepancies and suggests how the code should be modified.

3.4 TOOL ENVIRONMENT

Both the investigator and the verifier interact with shapes through a shared set of tools, while the generator produces code in a single LLM call without tool access. All tools are read-only which only serve for inspection and cannot modify the shapes.

Code execution. Both the investigator and verifier can execute arbitrary Python code against the shape via the CadQuery API. This is the most flexible tool in the suite and the one agents rely on most heavily, enabling targeted geometric queries that fixed utilities cannot capture, such as measuring the radius of a specific cylindrical face, checking whether two faces share the same normal, or computing the cross-section at a given height.

Renderer. For visual inspection, the rendering tool uses an object-centric orbit camera parameterized by polar angle θ, azimuth ϕ, and radius multiplier r, which auto-scales to the object's size when $r = 1$. This spherical parameterization gives the agent an intuitive way to request canonical

or arbitrary viewpoints using three parameters instead of the six needed for Cartesian camera placement (Yin et al., 2026). The investigator renders only $\mathcal{B}$, while the verifier renders $\mathcal{B}$ and $\hat{\mathcal{B}}$ side by side from matched viewpoints.

Global statistics. For quick bulk summaries, the statistics tool returns a structured overview of a shape's geometry: bounding box extents and dimensions, volume, surface area, center of mass, and number of solids. When the input is a point cloud $\mathcal{P}$, it reports the point count, centroid, and bounding box instead. The verifier can query both $\mathcal{B}$ and $\hat{\mathcal{B}}$ in a single call, making it easy to spot bulk discrepancies such as a volume mismatch or a shifted bounding box. It can also compute the IoU between the two shapes to track reconstruction progress numerically.

4 Experiments

4.1 Experimental Setup

Models. Our method utilizes off-the-shelf LLMs without fine-tuning. We evaluate two configurations: the open-source Qwen3.6-27B (Qwen Team, 2026) and the proprietary Gemini-3.1-Pro (Google DeepMind, 2025), referred to as *Ours-Qwen* and *Ours-Gemini* respectively.

Datasets. We evaluate on the test splits of DeepCAD (Wu et al., 2021) and Fusion 360 (Willis et al., 2021), as well as the ABC dataset (Koch et al., 2019). DeepCAD and Fusion 360 are dominated by geometrically simple shapes, such as plain boxes. Consequently, averaging metrics over the full test set can obscure performance differences on harder shapes. Following Sun et al. (2026), we use B-rep face count as a complexity proxy and partition shapes into Easy with fewer than 10 faces, Medium with 10–30 faces, and Hard with 30–60 faces. We randomly sample 50 shapes per group and report metrics separately. We cap the Hard group at 60 faces because shapes beyond this threshold are vanishingly rare in the standard benchmarks (e.g., only 9 of 8,009 DeepCAD test shapes exceed it). Appendix A.1 shows the corresponding face-count distributions.

Metrics. We report two complementary metrics. Volumetric Intersection over Union (IoU) measures the intersection over union of the predicted and target B-rep, computed directly on the B-rep geometry via Boolean intersection and union in CadQuery. Chamfer Distance (CD) (Fan et al., 2016) measures the symmetric mean nearest-neighbor distance between point clouds sampled from the surfaces of the shapes (8,192 points each). Before evaluation, both shapes are centered at the origin and scaled so that the longest bounding-box side equals a fixed reference length, ensuring scale- and translation-invariant comparison.

Baselines. We compare against three state-of-the-art CAD reverse engineering methods that take point clouds as input and output CadQuery code. *CAD-Recode* (Rukhovich et al., 2025) fine-tunes a 1.5B-parameter language model with a custom point cloud encoder to generate CadQuery programs in a single inference pass. *Cadrille* (Kolodiazhnyi et al., 2025) extends this approach to fine-tune a 2B-parameter model with multi-modal conditioning and online reinforcement learning. While these two methods perform reconstruction via a single inference pass, *CAD-Reasoner* (Kabisov et al., 2026) fine-tunes a VLM to iteratively rewrite CadQuery code over multiple refinement steps, conditioned on multi-view depth renderings and point cloud input. All three methods require task-specific training, whereas our approach uses off-the-shelf LLMs without fine-tuning.

Implementation Details. We reproduced all baseline results using a unified evaluation pipeline to ensure fair comparison across difficulty groups. For CAD-Recode we use point cloud input as it is the only supported modality. For Cadrille we use point cloud mode as well. For CAD-Reasoner we use multi-view depth input from B-rep. We evaluate CAD-Reasoner at both 5 and 30 refinement steps. Note that CAD-Reasoner renders the B-rep into multi-view depth images as model input, but the underlying source data is solid geometry, so we label it as *Solid* in Table 1.

For our system, the investigator is allowed at most 50 tool-use turns. We set the maximum generator step to $T{=}30$, and after each generator step, the verifier can perform up to 30 verification rounds. Notice that all these numbers are hard upper bounds where every agent can decide to terminate early.

Method	Input	DeepCAD			Fusion 360			ABC		
		Easy	Med.	Hard	Easy	Med.	Hard	Easy	Med.	Hard
Volumetric IoU ↑										
Cadrille	PC	0.936	0.901	0.708	0.901	0.789	0.649	0.879	0.773	0.544
CAD-Recode	PC	0.954	0.924	0.703	0.940	0.781	0.615	0.866	0.798	0.556
Ours (Qwen)	PC	0.866	0.921	0.797	0.970	0.797	0.664	0.907	0.820	0.650
Ours (Gemini)	PC	**0.998**	**0.990**	**0.905**	**0.998**	**0.955**	**0.843**	**0.967**	**0.961**	**0.805**
CAD-Reasoner	Solid	0.972	0.892	0.730	0.900	0.747	0.709	0.821	0.714	0.568
Ours (Qwen)	Solid	**1.000**	0.981	0.942	0.995	0.937	0.826	0.991	0.881	0.786
Ours (Gemini)	Solid	1.000	**1.000**	**0.980**	**1.000**	**0.968**	**0.967**	**0.997**	**0.981**	**0.950**
Chamfer Distance ↓										
Cadrille	PC	0.016	0.019	0.032	0.017	0.026	0.035	0.022	0.033	0.051
CAD-Recode	PC	0.015	0.017	0.030	0.015	0.032	0.043	0.024	0.035	0.056
Ours (Qwen)	PC	0.019	0.024	0.031	0.014	0.029	0.040	0.019	0.031	0.044
Ours (Gemini)	PC	**0.013**	**0.014**	**0.023**	**0.013**	**0.016**	**0.027**	**0.014**	**0.017**	**0.031**
CAD-Reasoner	Solid	0.014	0.023	0.029	0.018	0.034	0.029	0.031	0.047	0.048
Ours (Qwen)	Solid	**0.013**	0.015	0.020	0.013	0.024	0.031	0.015	0.031	0.031
Ours (Gemini)	Solid	0.013	**0.013**	**0.014**	**0.013**	**0.013**	**0.014**	**0.014**	**0.017**	**0.017**

Table 1: Quantitative results on DeepCAD, Fusion 360, and ABC across three complexity groups. We report Volumetric IoU (↑) and Chamfer Distance (↓). Input denotes the representation given to each method: PC = point cloud, Solid = closed watertight solid (B-rep or mesh). CAD-Reasoner converts B-rep to mesh and renders multi-view depth images as model input, but the source data is nonetheless solid geometry. Best results are in **bold**.

4.2 Main Results

Table 1 compares our method against baselines on all three datasets and three complexity levels.

B-rep input. With B-rep as input, both configurations of our method (*Ours-Qwen* and *Ours-Gemini*) substantially outperform the baselines on all splits. On the DeepCAD hard split, *Ours-Gemini* achieves an IoU of 0.980, surpassing the best baseline by 25 points, and attains perfect reconstruction (IoU = 1.0) on both the Easy and Medium splits. The trend is equally observable on Fusion 360, where *Ours-Gemini* reaches 0.967 IoU on the hard split compared to 0.709 (CAD-Reasoner) and 0.649 (Cadrille). On ABC, our method also maintains a large advantage across all difficulty levels. Even with the open-source Qwen3.6-27B, our method outperforms all baselines across all splits, demonstrating that the framework generalizes across LLMs, though stronger models yield better results. Chamfer Distance follows the same trend, confirming that the improvement is consistent across both volumetric and surface-level metrics. Figure 4 shows qualitative examples where baselines miss fine-grained details that our method recovers faithfully.

Point cloud input. Point-cloud input reduces performance because it discards exact geometry that the investigator and verifier can query from B-reps. Nevertheless, *Ours-Gemini* (PC) still outperforms all baselines on every Hard split, reaching 0.905 IoU on DeepCAD, 0.843 on Fusion 360, and 0.805 on ABC. *Ours-Qwen* (PC) surpasses the point-cloud baselines on most splits and exceeds CAD-Reasoner on most Hard splits despite CAD-Reasoner's richer solid-derived input, falling slightly behind only on Fusion 360 Hard (0.664 vs. 0.709). These results show that the agentic loop remains effective even with a less precise representation.

4.3 Multi-Turn Refinement

CAD reverse engineering benefits from multi-turn refinement. Figure 3 shows how mean IoU evolves over refinement steps on the Hard splits of all three datasets.

Our method refines more effectively per step than CAD-Reasoner. On Fusion 360 Hard, *Ours-Gemini* (B-rep) jumps from 0.68 IoU on the first attempt to 0.90 on the second, whereas CAD-Reasoner shows only modest improvement across its five steps and plateaus near the single-pass

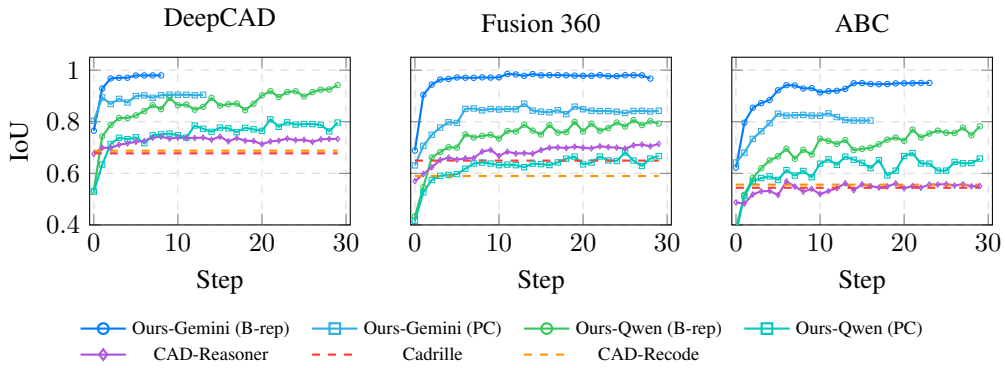


Figure 3: Mean IoU vs. refinement step on the Hard splits. Dashed lines indicate single-pass baselines (Cadrille, CAD-Recode). For samples that converge early, their final IoU is held constant for the remaining steps when computing the group mean. Some curves end before step 30 because every run in that group terminated in fewer than 30 steps.

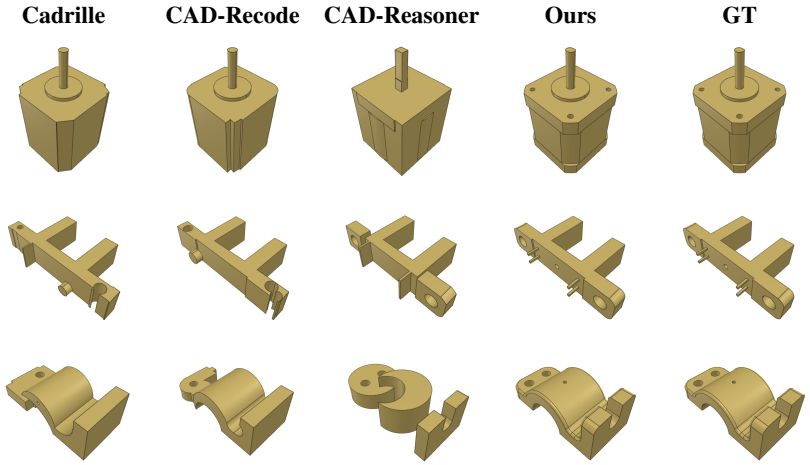


Figure 4: Qualitative comparison. From left to right: Cadrille, CAD-Recode, CAD-Reasoner, Ours, Ground Truth.

baselines. *Ours-Qwen* (B-rep) surpasses the single-pass baselines within five steps on every Hard split and overtakes CAD-Reasoner on DeepCAD and ABC Hard within that budget, reaching CAD-Reasoner's Fusion 360 performance by step 6, suggesting that programmatic inspection provides a much richer error signal than the visual feedback CAD-Reasoner conditions on.

The convergence profile also differs by input modality. B-rep input yields smooth, fast convergence, while point cloud input produces slower improvement with some oscillation, as point clouds discard the exact geometry that LLMs can query programmatically. Despite this, *Ours-Gemini* (PC) eventually outperforms all baselines on every Hard split.

Finally, refinement steps can partially compensate for model capability. *Ours-Gemini* (B-rep) converges within a few steps on DeepCAD Hard, whereas *Ours-Qwen* (B-rep) improves overall from 0.53 to 0.94 IoU over the full budget despite some oscillation. On ABC Hard, Qwen continues gaining through the later steps, suggesting that additional iterations can partially offset the difference in base model capability.

4.4 Ablation Studies

Table 2 reports ablation results on the Hard split of all three datasets using *Ours-Gemini* with both input modalities. We test an *Image-only* baseline and three configurations each removing one tool from the full system.

	B-rep input						*Point-cloud input*					
	DeepCAD		Fusion 360		ABC		DeepCAD		Fusion 360		ABC	
Method	IoU↑	CD↓	IoU↑	CD↓	IoU↑	CD↓	IoU↑	CD↓	IoU↑	CD↓	IoU↑	CD↓
Image-only	0.361	0.098	0.330	0.098	0.376	0.092	0.351	0.093	0.306	0.102	0.303	0.104
w/o Code inspection	0.751	0.037	0.573	0.054	0.641	0.047	0.619	0.050	0.550	0.064	0.553	0.060
w/o Global statistics	0.985	0.015	0.929	0.017	0.882	0.023	0.899	**0.022**	0.793	0.029	**0.854**	**0.024**
w/o Renderer	**0.999**	0.014	0.954	0.018	0.938	0.019	0.898	0.024	0.804	0.033	0.842	0.033
Ours (full)	0.980	**0.014**	**0.967**	**0.014**	**0.950**	**0.017**	**0.905**	0.023	**0.843**	**0.027**	0.805	0.031

Table 2: Ablation study on the Hard split with Gemini under B-rep and point-cloud inputs. The two input modalities are shown side by side, and each row removes one component from the full system. *Image-only* denotes that the investigator is dropped and the verifier has no access to ground-truth geometry.

The *Image-only* baseline directly isolates whether our gains come from tool access or simply from using a strong frontier model. Here the investigator is dropped and the generator and verifier see only renderings from six canonical angles, with no programmatic access to the geometry, while the model and iteration budget are otherwise unchanged. Performance collapses to an IoU of 0.33 to 0.38 across all splits and both modalities, far below the full system. Since the only removed variable is the ability to query the geometry, the gap is hard to attribute to the model's image understanding or iteration budget, and instead points to active geometry access as the main driver of reconstruction quality.

This is reinforced by the tool-removal configurations. Removing Python code inspection causes the largest drop, with IoU falling to 0.573 on Fusion 360 Hard and 0.641 on ABC Hard under B-rep input, confirming that targeted geometric queries are the most critical component. Removing global statistics or the renderer has a much smaller effect, indicating these tools provide complementary but less essential signal once code-based inspection is available. The renderer is the exception under point-cloud input, where its removal hurts more than under B-rep. We attribute this to the fact that B-rep allows exact geometric queries whereas point clouds do not, so rendered views provide a useful visual signal of the input, to complement the noisier programmatic queries. The contribution of the renderer thus scales with how much information is lost in the input representation.

5 Conclusion

We presented an agentic framework for CAD reverse engineering that reformulates 3D perception as an active, tool-based investigation, enabling frozen frontier LLMs to query geometry directly and iteratively reconstruct construction sequences without a trained 3D encoder or any task-specific fine-tuning. To our knowledge, this is the first system to bring a multi-agent loop with programmatic geometry access to CAD reverse engineering. Our method substantially outperforms all fine-tuned baselines across three datasets and difficulty levels, under both B-rep and point cloud inputs, demonstrating that multi-turn refinement with direct geometry access is a highly effective paradigm for this task.

The primary limitation of our method is inference cost. Each reconstruction requires multiple LLM calls over many refinement steps, averaging $0.2 per shape on Easy cases and $1.9 on Hard cases with Gemini-3.1-Pro, exceeding the cost of single-pass baselines. However, this cost differs in kind from fine-tuned approaches: those methods front-load a large one-time training cost that is amortized over many inference runs, whereas our method requires no training and pays only per shape. This tradeoff is well suited to settings that prioritize reconstruction quality over throughput, such as reverse engineering a modest number of complex parts, where there may be too few target parts to amortize the training cost of a dedicated model, and where the resulting model may still fall short of the required reconstruction quality. Our method is also not tied to proprietary models. The locally hostable Qwen model already performs strongly, offering a self-contained path for cost-sensitive or privacy-conscious settings. We expect inference cost to decrease through better termination criteria, improved tool design, and more capable or efficient LLMs, and hope this work encourages further exploration of agentic approaches for tasks requiring precise 3D reasoning.

REFERENCES

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating cad code with vision-language models for 3d designs. In *International conference on learning representations (ICLR)*, 2025.

Blender Online Community. Blender - a 3d modelling and rendering package, 2026. URL http://www.blender.org.

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, 2025. URL https://arxiv.org/abs/2504.04753.

CadQuery contributors. Cadquery, 2026. URL https://doi.org/10.5281/zenodo.18633916.

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. Transcad: A hierarchical transformer for cad sequence inference from point clouds, 2024. 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, 2026. URL https://arxiv.org/abs/2602.16317.

Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image, 2016. URL https://arxiv.org/abs/1612.00603.

Google DeepMind. Gemini 3.1 Pro. Model page, 2025. Retrieved from https://deepmind.google/models/gemini/pro/.

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

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

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, 2026. URL https://arxiv.org/abs/2603.29847.

Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. Ucsg-net – unsupervised discovering of constructive solid geometry tree, 2020. URL https://arxiv.org/abs/2006.09102.

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, 2024a. URL https://arxiv.org/abs/2402.17678.

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

Mingi Kim, Yongjun Kim, Jungwoo Kang, and Hyungki Kim. Brepcoder: A unified multimodal large language model for multi-task b-rep reasoning, 2026. URL https://arxiv.org/abs/2602.22284.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2019.

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 online reinforcement learning. *arXiv preprint arXiv:2505.22914*, 2025.

Joseph George Lambourne, Karl Willis, Pradeep Kumar Jayaraman, Longfei Zhang, Aditya Sanghi, and Kamal Rahimi Malekshan. Reconstructing editable prismatic cad from rounded voxel models. In *SIGGRAPH Asia 2022 Conference Papers*, SA '22, pp. 1–9. ACM, November 2022. doi: 10.1145/3550469.3555424. URL http://dx.doi.org/10.1145/3550469.3555424.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-llama: Leveraging large language models for computer-aided design parametric 3d model generation, 2025. URL https://arxiv.org/abs/2505.04481.

Pu Li, Jianwei Guo, Xiaopeng Zhang, and Dong ming Yan. Secad-net: Self-supervised cad reconstruction by learning sketch-extrude operations, 2023a. URL https://arxiv.org/abs/2303.10613.

Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Seek-cad: A self-refined generative modeling for 3d parametric cad using local inference via deepseek, 2026. URL https://arxiv.org/abs/2505.17702.

Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models, 2023b. URL https://arxiv.org/abs/2305.10355.

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 *Computer Vision and Pattern Recognition Conference*, 2024.

Dimitrios Mallis, Ahmet Serdar Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Maria Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-assistant: Tool-augmented vllms as generic cad task solvers. *ArXiv*, abs/2412.13810, 2024.

Nirmal Prasad Panta, Saugat Kafley, Rujal Acharya, Sashank Parajuli, Dikshya Parajuli, Prince Panta, Saroj Belbase, Sudikshya Pant, Amit Regmi, Akio Tanaka, and Christopher McComb. MEDA: A Multi-Agent System for Parametric CAD Model Creation. In *Proceedings of the ASME 2025 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE 2025)*, Anaheim, CA, USA, 2025. American Society of Mechanical Engineers (ASME). doi: 10.1115/DETC2025-163946. URL https://doi.org/10.1115/DETC2025-163946.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

Pooyan Rahmanzadehgervi, Logan Bolton, Mohammad Reza Taesiri, and Anh Totti Nguyen. Vision language models are blind: Failing to translate detailed visual features into words, 2025. URL https://arxiv.org/abs/2407.06581.

Daxuan Ren, Jianmin Zheng, Jianfei Cai, Jiatong Li, and Junzhe Zhang. Extrudenet: Unsupervised inverse sketch-and-extrude for shape parsing, 2022. URL https://arxiv.org/abs/2209.15632.

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.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Yuanxu Sun, Yuezhou Ma, Haixu Wu, Guanyang Zeng, Muye Chen, Jianmin Wang, and Mingsheng Long. Brep2shape: Boundary and shape representation alignment via self-supervised transformers, 2026. URL https://arxiv.org/abs/2602.07429.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. Eyes wide shut? exploring the visual shortcomings of multimodal llms, 2024. URL https://arxiv.org/abs/2401.06209.

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, 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, 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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(8):7880–7888, April 2025b. ISSN 2159-5399. doi: 10.1609/aaai.v39i8.32849. URL http://dx.doi.org/10.1609/aaai.v39i8.32849.

Kevin J. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Institute, 1986.

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, 2021. 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 (ICCV)*, pp. 6772–6782, October 2021.

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

Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm, 2025. 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, 2024. URL https://arxiv.org/abs/2308.16911.

Shaofeng Yin, Jiaxin Ge, Zora Zhiruo Wang, Chenyang Wang, Xiuyu Li, Michael J. Black, Trevor Darrell, Angjoo Kanazawa, and Haiwen Feng. Vision-as-inverse-graphics agent via interleaved multimodal reasoning, 2026. URL https://arxiv.org/abs/2601.11109.

Yang You, Mikaela Angelina Uy, Jiaqi Han, Rahul Thomas, Haotong Zhang, Yi Du, Hansheng Chen, Francis Engelmann, Suya You, and Leonidas Guibas. Img2cad: Reverse engineering 3d cad models from images through vlm-assisted conditional factorization, 2025. URL https://arxiv.org/abs/2408.01437.

Fenggen Yu, Qimin Chen, Maham Tanveer, Ali Mahdavi Amiri, and Hao Zhang. D^{2}CSG: Unsupervised learning of compact csg trees with dual complements and dropouts. *Advances in Neural Information Processing Systems*, 36, 2024.

Shuming Zhang, Zhidong Guan, Hao Jiang, Tao Ning, Xiaodong Wang, and Pingan Tan. Brep2seq: a dataset and hierarchical deep learning network for reconstruction and generation of computer-aided design models. *Journal of Computational Design and Engineering*, 11(1):110–134, 2024.

A Appendix

This appendix supplements the main paper with implementation details and additional results. A.1 shows the face-count distributions that motivate our stratified evaluation. A.2 reports per-task agent turns and API costs across models, input modalities, and difficulty levels. A.3 documents the system prompts and tool schemas given to each agent. A.4 provides full investigator interaction logs for the example in Figure 2. A.5 shows additional qualitative comparisons with the final CadQuery programs produced by our system.

A.1 Face-Count Distribution

Figure 5 shows the distribution of B-rep face counts across the evaluation datasets. DeepCAD and Fusion 360 concentrate at low face counts, whereas ABC covers a broader range of geometric complexity. We therefore report results separately across face-count groups.

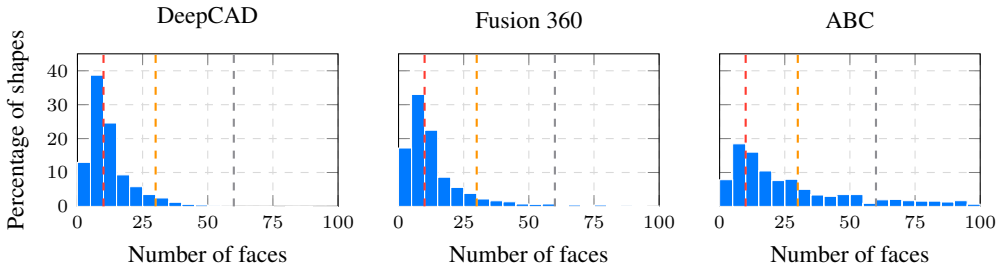


Figure 5: Distribution of B-rep face counts up to 100 on the DeepCAD and Fusion 360 test sets and a random subset of ABC. Dashed lines indicate the face-count thresholds (10, 30, and 60) used to define the evaluation groups.

A.2 Agent Turn and Cost Analysis

The agent turn and cost tables summarize 1800 task executions across two input modes (B-rep and point cloud), two base models (Gemini and Qwen), and three difficulty levels. Each input/model setting contains 450 tasks, with 150 Easy, 150 Medium, and 150 Hard tasks. The split is balanced across datasets: DeepCAD, Fusion 360 and ABC. We count one agent turn as one model call. Dollar costs use Gemini pricing of $2.00/M input tokens and $12.00/M output tokens. Qwen dollar costs are omitted because it was hosted locally.

Input	Model	Diff.	Turns (med./mean)	Input MTok	Output MTok	Cost/task (med./mean)
B-rep	Gemini	Easy	12.0/17.0	0.081	0.003	$0.11/$0.20
B-rep	Gemini	Med.	20.0/39.1	0.292	0.011	$0.27/$0.72
B-rep	Gemini	Hard	41.5/81.9	0.782	0.028	$0.79/$1.90
B-rep	Qwen	Easy	19.0/37.3	0.389	0.054	–
B-rep	Qwen	Med.	57.0/147.0	2.390	0.319	–
B-rep	Qwen	Hard	273.0/280.9	4.921	0.591	–
Point cloud	Gemini	Easy	22.0/33.0	0.237	0.011	$0.34/$0.60
Point cloud	Gemini	Med.	49.5/113.4	1.164	0.046	$1.22/$2.88
Point cloud	Gemini	Hard	127.0/175.9	1.909	0.071	$3.41/$4.67
Point cloud	Qwen	Easy	36.5/93.1	1.263	0.151	–
Point cloud	Qwen	Med.	99.5/230.0	3.718	0.465	–
Point cloud	Qwen	Hard	409.0/346.3	5.964	0.718	–

Table 3: Agent turn and token/cost summary by difficulty, input mode, and base model. Turns are model calls summed across the investigator, generator, and verifier agents. Token columns report mean millions of tokens per task. Dollar costs are reported only where pricing is specified.

Input	Model	Diff.	Turns per task (med./mean) Inv.	Gen.	Ver.	Token share (%) Inv.	Gen.	Ver.
B-rep	Gemini	Easy	7.0/8.9	2.0/2.6	1.0/5.6	43.4	12.4	44.2
B-rep	Gemini	Med.	13.0/16.5	3.0/3.6	2.0/18.9	35.0	10.8	54.2
B-rep	Gemini	Hard	22.5/27.9	3.0/6.3	13.5/47.6	27.5	10.4	62.1
B-rep	Qwen	Easy	13.0/15.9	3.0/5.4	2.0/16.0	25.8	16.8	57.4
B-rep	Qwen	Med.	26.0/29.0	7.0/18.6	18.0/99.4	11.1	21.3	67.5
B-rep	Qwen	Hard	36.5/36.3	27.5/31.5	207.5/213.2	7.3	20.6	72.1
Point cloud	Gemini	Easy	12.0/15.1	2.0/3.0	4.0/14.9	43.4	7.6	49.0
Point cloud	Gemini	Med.	23.0/27.6	3.0/5.8	19.0/80.0	22.4	8.2	69.5
Point cloud	Gemini	Hard	42.0/36.3	5.0/8.7	82.5/130.9	19.4	8.4	72.3
Point cloud	Qwen	Easy	20.0/26.1	4.0/9.8	11.0/57.2	21.9	12.5	65.6
Point cloud	Qwen	Med.	34.0/35.3	10.0/19.6	49.5/175.1	11.5	11.9	76.7
Point cloud	Qwen	Hard	51.0/42.3	32.0/29.1	321.0/275.0	9.0	13.3	77.7

Table 4: Role-level decomposition of agent turns and token use. Token shares are computed from total tokens consumed by each agent role over the split. Inv., Gen., and Ver. denote investigator, generator, and verifier, respectively.

A.3 Agent Prompts and Tools

A.3.1 System Prompts

The following system prompts define the role and response format for each agent.

Investigator

```
[Role]
You are CadQueryInvestigator --- an expert geometric analyst of 3D CAD shapes. You receive
a target CAD shape and optional reference images. Your sole task is to thoroughly
investigate the geometry using your available tools, then produce a precise, actionable
reconstruction plan for the CadQueryGenerator.

[Response Format]
The task proceeds over multiple rounds. In each round, your response must be exactly one
tool call with reasoning in the content field. If you would like to call multiple tools,
call them one by one in successive turns. Always return both the tool call and the content
together in one response.
```

Generator

```
[Role]
You are CadQueryGenerator --- an expert, tool-driven agent that writes CadQuery Python code
to reconstruct a target 3D solid. You will receive a target description and/or reference
images of the target shape. Your goal is to reproduce it as faithfully as possible.
Each round you will also receive feedback from the verifier identifying discrepancies and
recommending next steps. Incorporate that feedback carefully into your next code revision.

[Response Format]
The task proceeds over multiple rounds. In each round, your response must be exactly one
tool call with reasoning in the content field. If you would like to call multiple tools,
call them one by one in successive turns. Always return both the tool call and the content
together in one response.

[CadQuery execution]
Your script runs headless (no viewer). Do not call show_object, show(), or any
display/visualization helpers. Build geometry only. You must store the final CadQuery shape
(Workplane or Shape) in a variable named r (exactly r, no other name for the final result).
Do not add export, render, or show code.

[Alignment & metrics]
Match the target's placement and orientation in world space so multi-view renders overlap
the reference. **Align your model using the exact Bounding Box limits (X min/max, Y
min/max, Z min/max) rather than the Center of Mass**, as Center of Mass is highly sensitive
to minor internal thickness errors; wrong pose tanks IoU even if the part shape is right.
Prioritize silhouette and major feature layout for visual similarity, preserve distinctive
holes, fillets, and proportions for feature similarity, and tighten occupied volume
agreement to raise render-based similarity and Intersection over Union (IoU).
```

Verifier

```
[Role]
You are CadQueryVerifier --- an expert reviewer of 3D CAD shapes. Each round you receive
the CadQuery code produced by the generator, along with a target description and/or
reference images of the target shape.
Use your available tools as needed to inspect the current shape and make a thorough
assessment.
Your task is to precisely and comprehensively identify discrepancies between the current
shape and the target, then propose clear, actionable recommendations for the generator.

[Response Format]
The task proceeds over multiple rounds. In each round, your response must be exactly one
tool call with reasoning in the content field. If you would like to call multiple tools,
call them one by one in successive turns. Always return both the tool call and the content
together in one response.

[Alignment & metrics]
Flag translation/rotation/mirror errors versus the target early; misalignment dominates
failed IoU. Advise the generator to align the object using the Bounding Box limits (X
min/max, Y min/max, Z min/max) rather than Center of Mass, as the center of mass is highly
sensitive to internal thickness errors. In feedback, tie issues to views: silhouette
mismatch (visual similarity), missing or wrong features (feature similarity), and
bulk/volume disagreement (overlap / IoU). Ask the generator to fix pose before micro-tuning
details when renders clearly miss the reference. Always frame feedback in terms of how each
fix will improve IoU --- the primary optimization target is to maximize IoU.
```

A.3.2 Tool Schema

Each agent is equipped with a set of tools. Below we list the tools available to each agent, along with their arguments and full descriptions as seen by the model.

Investigator Tools

`render_gt`	
Arguments	`polar_angle`, `azimuth`, `radial`
Description	Render the ground-truth from a given orbit camera pose and return the saved image path. If the GT was provided as a .step file, it is rendered as a solid mesh; if it was provided as a point cloud (.ply/.npy), it is rendered as a point cloud. The camera always looks at the origin (object-centric). An axis triad is visible in the bottom-left corner (Red=X, Green=Y, Blue=Z) to help you determine orientation. Examples: (polar_angle=0, azimuth=0) -> top view (looking down -Z, image right is +X, image up is +Y); (polar_angle=90, azimuth=0) -> front view (looking from -Y, image right is +X, image up is +Z); (polar_angle=90, azimuth=90) -> right view (looking from +X, image right is +Y, image up is +Z); (polar_angle=90, azimuth=180) -> back view (looking from +Y, image right is -X, image up is +Z); (polar_angle=90, azimuth=270) -> left view (looking from -X, image right is -Y, image up is +Z); (polar_angle=180, azimuth=0) -> bottom view (looking from -Z, image right is +X, image up is -Y).

`get_gt_stats`	
Arguments	—
Description	Return global geometry statistics for the GT. If GT is a .step file, reports volume, area, bbox, topology counts, and face/edge type histograms. If GT is a point cloud (.ply/.npy), reports n_points, centroid, bbox, mean_radius, and max_radius.

`execute_inspection_code_gt`	
Arguments	`thought`, `code`
Description	Execute arbitrary Python code to inspect the ground-truth target. The namespace is mode-aware: - GT is a .step file: `cq` and `gt_shape` (cq.Shape) are available. - GT is a point cloud (.ply/.npy): `np` and `gt_points` (NumPy (N,3) array) are available. Use this to programmatically measure properties of the ground-truth (volume, bbox, face count, point-set statistics, etc). Print results with print() or assign to `result`. This tool is for observation only; if you need to modify the CAD model, end verification and hand back to the generator. Return everything through print(). Do NOT write files (no plt.savefig, no Image.save, no open(..., 'w'));

`end`	
Arguments	`overall_description`, `detailed_plan`
Description	Call this tool when your geometric investigation is complete. Output a shape description and geometric reconstruction plan. Do not call this until you have rendered the shape from multiple views and identified all major geometric features.

Generator Tools

`execute_and_evaluate`	
Arguments	`thought`, `code_diff`, `code`
Description	Execute CadQuery Python code and render the resulting shape. Returns either: (1) On error: the traceback and stdout so you can diagnose and fix the issue; or (2) On success: a multi-view render of the shape and feedback from a verifier agent. Rules for your code: - Use the `cadquery` library (import cadquery as cq). - Assign the final shape to a variable named `r` (a cq.Workplane or cq.Shape). Do NOT add any export calls yourself — the executor handles that. - Do not include rendering or display calls.

Verifier Tools

`render_pred_vs_gt`	
Arguments	`polar_angle`, `azimuth`, `radial`, `pred_as`
Description	Render the predicted CAD shape alongside the ground-truth from a given orbit camera pose and return the saved image path. The two are placed side-by-side with labels. If the GT was provided as a point cloud (.ply/.npy), it is rendered as a point cloud; the prediction is always a CAD shape but can optionally be sampled to a point cloud via ``pred_as='points'`` for a like-for-like visual comparison. The view angles for both sides are the same, but their absolute positions may be different. The camera always looks at the origin (object-centric). An axis triad is visible in the bottom-left corner (Red=X, Green=Y, Blue=Z) to help you determine orientation. Examples: (polar_angle=0, azimuth=0) -> top view (looking down -Z, image right is +X, image up is +Y); (polar_angle=90, azimuth=0) -> front view (looking from -Y, image right is +X, image up is +Z); (polar_angle=90, azimuth=90) -> right view (looking from +X, image right is +Y, image up is +Z); (polar_angle=90, azimuth=180) -> back view (looking from +Y, image right is -X, image up is +Z); (polar_angle=90, azimuth=270) -> left view (looking from -X, image right is -Y, image up is +Z); (polar_angle=180, azimuth=0) -> bottom view (looking from -Z, image right is +X, image up is -Y).

`get_pred_and_gt_stats`	
Arguments	—
Description	Return global statistics for both the predicted shape and the GT. The pred side is always shape-style stats; the GT side is shape-style when GT is a .step and point-cloud-style when GT is a .ply/.npy.

`get_iou`	
Arguments	—
Description	Compute an Intersection-over-Union score between the predicted CAD shape and the ground-truth target. The metric depends on how the GT was provided: - GT is a CAD shape (.step): volumetric IoU is reported in two variants: - raw_iou: IoU computed on the shapes in their original coordinate frames. - aligned_iou: IoU after independently centering each shape at the origin and scaling so the longest bounding-box side matches, removing position and scale differences. - GT is a point cloud (.ply/.npy): the predicted shape is surface-sampled and compared against the GT points using voxel IoU (raw and GT-normalized) plus the symmetric Chamfer distance (raw and GT-normalized). Voxel IoU of 1.0 means perfect occupancy overlap at the chosen resolution; 0.0 means no overlap.

execute_inspection_code	
Arguments	thought, code
Description	Execute arbitrary Python code to inspect the predicted CAD shape and the ground truth. The namespace always includes `cq` and `pred_shape` (a cq.Shape). The GT is exposed based on how it was provided: - GT is a .step file: `gt_shape` is available as a cq.Shape. - GT is a point cloud (.ply/.npy): `gt_points` is available as a NumPy array of shape (N, 3); `np` is pre-imported. Use this to programmatically measure differences: volume, bounding box, face count, surface area, topology, vertex positions, point-set statistics, etc. Print results with print() or assign to `result` for structured output. This tool is for observation only; if you need to modify the CAD model, end verification and hand back to the generator. Return everything through print(). Do NOT write files (no plt.savefig, no Image.save, no open(..., 'w'));

end	
Arguments	visual_difference, edit_suggestion
Description	If you think your observations are sufficient, call the tool to end the process and output the answer.

A.4 Investigator Agent Transcripts

We provide the complete investigator interaction histories for the example in Figure 2, for both B-rep and point-cloud inputs.

A.4.1 B-rep Input

System

```
[Role]
You are CadQueryInvestigator --- an expert geometric analyst of 3D CAD shapes. You receive
a target CAD shape. Your sole task is to thoroughly investigate the geometry using your
available tools, then produce a precise, actionable reconstruction plan for the
CadQueryGenerator.

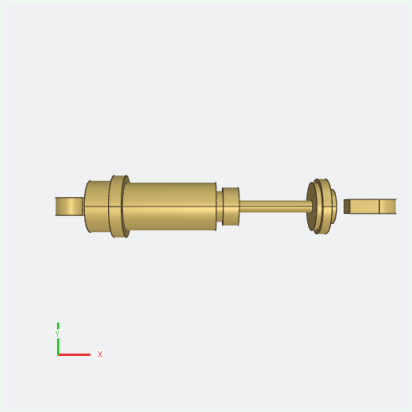
[Response Format]
The task proceeds over multiple rounds. In each round, your response must be exactly one
tool call with reasoning in the content field. If you would like to call multiple tools,
call them one by one in successive turns. Always return both the tool call and the content
together in one response.
```

Agent → render_gt

```
{
  "azimuth": 0,
  "polar_angle": 0,
  "radial": 1
}
```

Output (render_gt)

Render gt from view polar_angle=0.0, azimuth=0.0, radial=1.0.

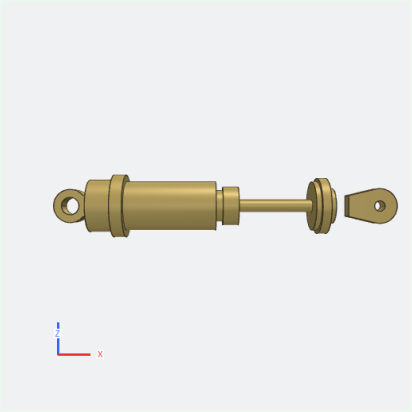


Agent → `render_gt`

```
{
  "azimuth": 0,
  "polar_angle": 90,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=90.0, azimuth=0.0, radial=1.0.

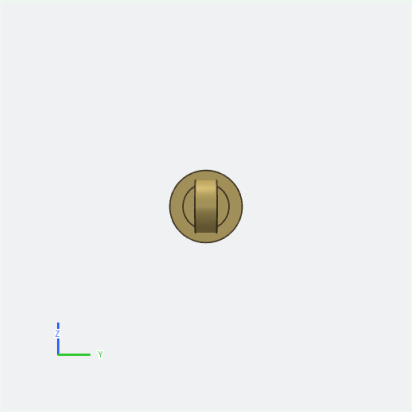


Agent → `render_gt`

```
{
  "azimuth": 90,
  "polar_angle": 90,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=90.0, azimuth=90.0, radial=1.0.



Agent → execute_inspection_code_gt

```
print(f"Number of solids: {len(gt_shape.Solids())}")
print(f"Bounding box: {gt_shape.BoundingBox()}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Number of solids: 2
Bounding box: <cadquery.occ_impl.geom.BoundBox object at 0x147347500a90>
```

Agent → execute_inspection_code_gt

```
solids = gt_shape.Solids()
print(f"Number of solids: {len(solids)}")
for i, s in enumerate(solids):
    bbox = s.BoundingBox()
    print(f"Solid {i} bbox: xmin={bbox.xmin:.4f}, xmax={bbox.xmax:.4f},
    ymin={bbox.ymin:.4f}, ymax={bbox.ymax:.4f}, zmin={bbox.zmin:.4f},
    zmax={bbox.zmax:.4f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Number of solids: 2
Solid 0 bbox: xmin=-4.7500, xmax=75.6500, ymin=-9.0000, ymax=9.0000, zmin=-9.0000,
zmax=9.0000
Solid 1 bbox: xmin=80.0000, xmax=93.9000, ymin=-2.0000, ymax=2.0000, zmin=-5.0000,
zmax=5.0000
```

Agent → execute_inspection_code_gt

```
solid0 = gt_shape.Solids()[0]
cyls = [f for f in solid0.Faces() if f.geomType() == 'CYLINDER']
for c in cyls:
    print(c.geomType(), "Radius:", getattr(c._geomAdaptor(), 'Cylinder', lambda:
    None)().Radius(),
          "Center:", c.Center().x, c.Center().y, c.Center().z)
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: CYLINDER Radius: 5.0 Center: 74.65
1.5234270863689947e-16 1.7763568394002505e-15
CYLINDER Radius: 7.0 Center: 70.4 -1.021206784761456e-17 0.0
CYLINDER Radius: 8.0 Center: 72.4 1.6563373569580392e-16 0.0
CYLINDER Radius: 1.75 Center: 58.95 -2.5530169619038866e-18 0.0
CYLINDER Radius: 5.5 Center: 46.0 3.0545910945659445e-16 -8.881784197001252e-16
CYLINDER Radius: 4.5 Center: 42.75 3.2494644847239465e-18 -8.881784197001252e-16
CYLINDER Radius: 7.0 Center: 28.749999999999996 -1.021206784761456e-17
-8.881784197001252e-16
CYLINDER Radius: 9.0 Center: 14.0 6.498928969445921e-18 -1.7763568394002505e-15
CYLINDER Radius: 7.5 Center: 8.75 -6.037521991065458e-17 -1.7763568394002505e-15
CYLINDER Radius: 2.5 Center: -8.881784197001252e-16 -7.774689221355262e-17
-7.617135431845028e-17
```

```
CYLINDER Radius: 4.75 Center: -1.2205022059653694 -4.27904097022986e-17
-7.407643079347305e-09
```

Agent → execute_inspection_code_gt

```
import numpy as np
solid0 = gt_shape.Solids()[0]
cyls = [f for f in solid0.Faces() if f.geomType() == 'CYLINDER']
for c in cyls:
    r = getattr(c._geomAdaptor(), 'Cylinder', lambda: None)().Radius()
    bb = c.BoundingBox()
    dir = getattr(c._geomAdaptor(), 'Cylinder', lambda: None)().Axis().Direction()
    axis = (dir.X(), dir.Y(), dir.Z())
    print(f"R={r:.3f}, X:[{bb.xmin:.3f}, {bb.xmax:.3f}], Y:[{bb.ymin:.3f}, {bb.ymax:.3f}],
    Z:[{bb.zmin:.3f}, {bb.zmax:.3f}], Axis={axis}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: R=5.000, X:[73.650, 75.650],
Y:[-5.000, 5.000], Z:[-5.000, 5.000], Axis=(1.0, 0.0, 0.0)
R=7.000, X:[69.650, 71.150], Y:[-7.000, 7.000], Z:[-7.000, 7.000], Axis=(-1.0, 0.0, 0.0)
R=8.000, X:[71.150, 73.650], Y:[-8.000, 8.000], Z:[-8.000, 8.000], Axis=(-1.0, 0.0, 0.0)
R=1.750, X:[48.250, 69.650], Y:[-1.750, 1.750], Z:[-1.750, 1.750], Axis=(1.0, 0.0, 0.0)
R=5.500, X:[43.750, 48.250], Y:[-5.500, 5.500], Z:[-5.500, 5.500], Axis=(1.0, 0.0, 0.0)
R=4.500, X:[41.750, 43.750], Y:[-4.500, 4.500], Z:[-4.500, 4.500], Axis=(1.0, 0.0, 0.0)
R=7.000, X:[15.750, 41.750], Y:[-7.000, 7.000], Z:[-7.000, 7.000], Axis=(1.0, 0.0, 0.0)
R=9.000, X:[12.250, 15.750], Y:[-9.000, 9.000], Z:[-9.000, 9.000], Axis=(1.0, 0.0, 0.0)
R=7.500, X:[5.250, 12.250], Y:[-7.500, 7.500], Z:[-7.500, 7.500], Axis=(1.0, 0.0, 0.0)
R=2.500, X:[-2.500, 2.500], Y:[-2.500, 2.500], Z:[-2.500, 2.500], Axis=(0.0, 1.0, 0.0)
R=4.750, X:[-4.750, 3.683], Y:[-2.500, 2.500], Z:[-4.750, 4.750], Axis=(0.0, 1.0, 0.0)
```

Agent → execute_inspection_code_gt

```
solid0 = gt_shape.Solids()[0]
planes = [f for f in solid0.Faces() if f.geomType() == 'PLANE']
for p in planes:
    dir = getattr(p._geomAdaptor(), 'Plane', lambda: None)().Axis().Direction()
    axis = (dir.X(), dir.Y(), dir.Z())
    bb = p.BoundingBox()
    print(f"Axis={axis}, X:[{bb.xmin:.3f}, {bb.xmax:.3f}], Y:[{bb.ymin:.3f},
    {bb.ymax:.3f}], Z:[{bb.zmin:.3f}, {bb.zmax:.3f}]")
```

Output (execute_inspection_code_gt)

```
Output:
Error: Traceback (most recent call last):
  File ".../4.py", line 6, in <module>
    dir = getattr(p._geomAdaptor(), 'Plane', lambda: None)().Axis().Direction()
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'Axis'
```

Agent → execute_inspection_code_gt

```
solid0 = gt_shape.Solids()[0]
planes = [f for f in solid0.Faces() if f.geomType() == 'PLANE']
for p in planes:
    try:
        norm = p.normalAt(p.Center())
        axis = (norm.x, norm.y, norm.z)
        bb = p.BoundingBox()
        print(f"Norm=({axis[0]:.2f}, {axis[1]:.2f}, {axis[2]:.2f}), X:[{bb.xmin:.3f},
        {bb.xmax:.3f}], Y:[{bb.ymin:.3f}, {bb.ymax:.3f}], Z:[{bb.zmin:.3f},
        {bb.zmax:.3f}]")
    except Exception as e:
        print("Error on a plane", e)
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Norm=(1.00, 0.00, 0.00), X:[75.650,
75.650], Y:[-5.000, 5.000], Z:[-5.000, 5.000]
Norm=(-1.00, 0.00, 0.00), X:[69.650, 69.650], Y:[-7.000, 7.000], Z:[-7.000, 7.000]
Norm=(-1.00, -0.00, -0.00), X:[71.150, 71.150], Y:[-8.000, 8.000], Z:[-8.000, 8.000]
Norm=(1.00, 0.00, 0.00), X:[73.650, 73.650], Y:[-8.000, 8.000], Z:[-8.000, 8.000]
Norm=(-1.00, -0.00, -0.00), X:[43.750, 43.750], Y:[-5.500, 5.500], Z:[-5.500, 5.500]
```

```
Norm=(1.00, 0.00, 0.00), X:[48.250, 48.250], Y:[-5.500, 5.500], Z:[-5.500, 5.500]
Norm=(1.00, 0.00, 0.00), X:[41.750, 41.750], Y:[-7.000, 7.000], Z:[-7.000, 7.000]
Norm=(-1.00, -0.00, -0.00), X:[12.250, 12.250], Y:[-9.000, 9.000], Z:[-9.000, 9.000]
Norm=(1.00, 0.00, 0.00), X:[15.750, 15.750], Y:[-9.000, 9.000], Z:[-9.000, 9.000]
Norm=(-1.00, -0.00, -0.00), X:[5.250, 5.250], Y:[-7.500, 7.500], Z:[-7.500, 7.500]
Norm=(0.00, -1.00, -0.00), X:[-4.750, 5.250], Y:[-2.500, -2.500], Z:[-4.750, 4.750]
Norm=(0.00, 0.00, -1.00), X:[3.683, 5.250], Y:[-2.500, 2.500], Z:[-3.000, -3.000]
Norm=(0.00, 0.00, 1.00), X:[3.683, 5.250], Y:[-2.500, 2.500], Z:[3.000, 3.000]
Norm=(-0.00, 1.00, 0.00), X:[-4.750, 5.250], Y:[2.500, 2.500], Z:[-4.750, 4.750]
```

Agent → execute_inspection_code_gt

```python
solid0 = gt_shape.Solids()[0]
other_faces = [f for f in solid0.Faces() if f.geomType() not in ('PLANE', 'CYLINDER')]
print(f"Number of other faces in Solid 0: {len(other_faces)}")
for f in other_faces:
    print(f.geomType())
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Number of other faces in Solid 0: 0
```

Agent → execute_inspection_code_gt

```python
solid1 = gt_shape.Solids()[1]
for f in solid1.Faces():
    geom = f.geomType()
    bb = f.BoundingBox()
    if geom == 'CYLINDER':
        r = getattr(f._geomAdaptor(), 'Cylinder', lambda: None)().Radius()
        dir = getattr(f._geomAdaptor(), 'Cylinder', lambda: None)().Axis().Direction()
        axis = (dir.X(), dir.Y(), dir.Z())
        print(f"CYLINDER R={r:.3f}, Axis={axis}, X:[{bb.xmin:.3f}, {bb.xmax:.3f}],
        Y:[{bb.ymin:.3f}, {bb.ymax:.3f}], Z:[{bb.zmin:.3f}, {bb.zmax:.3f}]")
    elif geom == 'PLANE':
        norm = f.normalAt(f.Center())
        axis = (norm.x, norm.y, norm.z)
        print(f"PLANE Norm=({axis[0]:.2f}, {axis[1]:.2f}, {axis[2]:.2f}), X:[{bb.xmin:.3f},
        {bb.xmax:.3f}], Y:[{bb.ymin:.3f}, {bb.ymax:.3f}], Z:[{bb.zmin:.3f},
        {bb.zmax:.3f}]")
    else:
        print(f"{geom}, X:[{bb.xmin:.3f}, {bb.xmax:.3f}], Y:[{bb.ymin:.3f}, {bb.ymax:.3f}],
        Z:[{bb.zmin:.3f}, {bb.zmax:.3f}]")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: PLANE Norm=(0.00, -1.00, -0.00),
X:[80.000, 93.900], Y:[-2.000, -2.000], Z:[-5.000, 5.000]
CYLINDER R=1.500, Axis=(0.0, 1.0, 0.0), X:[87.400, 90.400], Y:[-2.000, 2.000], Z:[-1.500,
1.500]
PLANE Norm=(-1.00, 0.00, 0.00), X:[80.000, 80.000], Y:[-2.000, 2.000], Z:[-3.000, 3.000]
PLANE Norm=(-0.23, 0.00, -0.97), X:[80.000, 87.730], Y:[-2.000, 2.000], Z:[-4.861, -3.000]
CYLINDER R=5.000, Axis=(0.0, 1.0, 0.0), X:[87.730, 93.900], Y:[-2.000, 2.000], Z:[-5.000,
5.000]
PLANE Norm=(-0.23, 0.00, 0.97), X:[80.000, 87.730], Y:[-2.000, 2.000], Z:[3.000, 4.861]
PLANE Norm=(-0.00, 1.00, 0.00), X:[80.000, 93.900], Y:[2.000, 2.000], Z:[-5.000, 5.000]
```

Agent → execute_inspection_code_gt

```python
solid0 = gt_shape.Solids()[0]
for p in [f for f in solid0.Faces() if f.geomType() == 'PLANE']:
    norm = p.normalAt(p.Center())
    if abs(norm.x) > 0.99:
        bb = p.BoundingBox()
        print(f"Norm_X={norm.x:.0f}, X={bb.xmin:.3f}, Y:[{bb.ymin:.3f}, {bb.ymax:.3f}],
        Z:[{bb.zmin:.3f}, {bb.zmax:.3f}]")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Norm_X=1, X=75.650, Y:[-5.000,
5.000], Z:[-5.000, 5.000]
Norm_X=-1, X=69.650, Y:[-7.000, 7.000], Z:[-7.000, 7.000]
Norm_X=-1, X=71.150, Y:[-8.000, 8.000], Z:[-8.000, 8.000]
Norm_X=1, X=73.650, Y:[-8.000, 8.000], Z:[-8.000, 8.000]
```

```
Norm_X=-1, X=43.750, Y:[-5.500, 5.500], Z:[-5.500, 5.500]
Norm_X=1, X=48.250, Y:[-5.500, 5.500], Z:[-5.500, 5.500]
Norm_X=1, X=41.750, Y:[-7.000, 7.000], Z:[-7.000, 7.000]
Norm_X=-1, X=12.250, Y:[-9.000, 9.000], Z:[-9.000, 9.000]
Norm_X=1, X=15.750, Y:[-9.000, 9.000], Z:[-9.000, 9.000]
Norm_X=-1, X=5.250, Y:[-7.500, 7.500], Z:[-7.500, 7.500]
```

Agent → execute_inspection_code_gt

```
print(f"Volume: {gt_shape.Volume():.2f}")
print(f"Solid 0 Volume: {gt_shape.Solids()[0].Volume():.2f}")
print(f"Solid 1 Volume: {gt_shape.Solids()[1].Volume():.2f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Volume: 8481.07
Solid 0 Volume: 8062.83
Solid 1 Volume: 418.24
```

Agent → end

```
Overall Description
The target CAD model is a mechanical assembly resembling a linear actuator, gas strut, or
shock absorber. It consists of two distinctly separate solid bodies aligned coaxially along
the primary axis (X-axis).

The larger main body (Solid 0) accounts for most of the overall mass (volume ~8062,
bounding box length ~80.4 units) and exhibits perfect rotational symmetry along most of its
length. Its left end features a sturdy mounting eyelet (with a radius 2.5 through-hole and
top/bottom flats). Extending to the right from the eyelet is a long, multi-stepped
cylindrical housing. This housing starts with a wider collar, steps down to a long main
barrel, and further necks down before transitioning into a long, thin piston rod (radius
1.75). The right end of the rod features three sequential disk-like caps/bosses of varying
diameters.

Positioned slightly to the right of the main body's terminal boss, across a 4.35-unit gap,
is a completely independent end attachment (Solid 1). This separate piece (volume ~418,
bounding box length ~13.9 units) is an eyelet/clevis tab. It has a flat front face pointing
towards the piston rod, tapering outwards tangentially to meet a rounded back end that
contains a small mounting hole (radius 1.5).

Both bodies feature sharp edges and precise geometric transitions. The entire object
exhibits strict structural alignment, designed to connect between two pivot points using
the outer eyelet holes.

Detailed Reconstruction Plan
1. Base Shape:
The object is a mechanical assembly resembling a shock absorber or gas strut. It consists
of two separate solid bodies aligned along the X-axis.
- Solid 0 (Main Body) spans from X = -4.75 to X = 75.65 and is primarily a series of
coaxial cylindrical segments with a mounting eyelet on the left end.
- Solid 1 (End Attachment) is a separate clevis/eyelet located to the right, spanning from
X = 80 to X = 93.9. There is a 4.35-unit gap between the two bodies.

2. Major Features (Solid 0 - Main Body):
- Left Mounting Eyelet: Centered at X = 0, Z = 0. Extruded symmetrically along the Y-axis
with a total thickness of 5 (from Y = -2.5 to 2.5). The outer profile is a cylinder of
radius 4.75, with a concentric through-hole of radius 2.5. The top and bottom are flattened
by horizontal planes at Z = 3 and Z = -3, which extend rightwards to connect to the
cylindrical body at X = 5.25.
- Cylindrical Housing (revolved or extruded along the X-axis):
  - X = 5.25 to 12.25: Radius 7.5 (Length = 7)
  - X = 12.25 to 15.75: Radius 9.0 (Length = 3.5, thick collar)
  - X = 15.75 to 41.75: Radius 7.0 (Length = 26, main body)
  - X = 41.75 to 43.75: Radius 4.5 (Length = 2, neck)
  - X = 43.75 to 48.25: Radius 5.5 (Length = 4.5)
- Piston Rod & End Caps (revolved or extruded along the X-axis):
  - X = 48.25 to 69.65: Radius 1.75 (Length = 21.4, thin piston rod)
  - X = 69.65 to 71.15: Radius 7.0 (Length = 1.5, washer/cap)
  - X = 71.15 to 73.65: Radius 8.0 (Length = 2.5, washer/cap)
  - X = 73.65 to 75.65: Radius 5.0 (Length = 2, terminal boss)

3. Major Features (Solid 1 - End Attachment):
- The right end attachment is an independent body extruded symmetrically along the Y-axis
with a thickness of 4 (from Y = -2 to 2).
- Profile on the XZ plane:
```

```
  - A vertical flat face at the front (left) located at X = 80, spanning from Z = -3 to Z =
  3.
  - A cylindrical outer arc of radius 5.0 centered at X = 88.9, Z = 0.
  - Straight sloped lines connecting the top and bottom edges of the vertical face (X = 80,
  Z = +/-3) tangentially to the cylindrical arc.
  - A central through-hole of radius 1.5, also centered at X = 88.9, Z = 0.

4. Surface Details:
- The model is composed entirely of perfect geometric primitives (cylinders and planar
extrusions).
- Transitions between the cylindrical sections on the main body are perfectly sharp flat
annular faces (no chamfers or fillets).
- The entire assembly is strictly symmetric about the XZ and XY planes.
```

A.4.2 Point Cloud Input

System

```
[Role]
You are CadQueryInvestigator --- an expert geometric analyst of 3D CAD shapes. You receive
a target CAD shape. Your sole task is to thoroughly investigate the geometry using your
available tools, then produce a precise, actionable reconstruction plan for the
CadQueryGenerator.

[Response Format]
The task proceeds over multiple rounds. In each round, your response must be exactly one
tool call with reasoning in the content field. If you would like to call multiple tools,
call them one by one in successive turns. Always return both the tool call and the content
together in one response.
```

Agent → `get_gt_stats`

```
{}
```

Output (`get_gt_stats`)

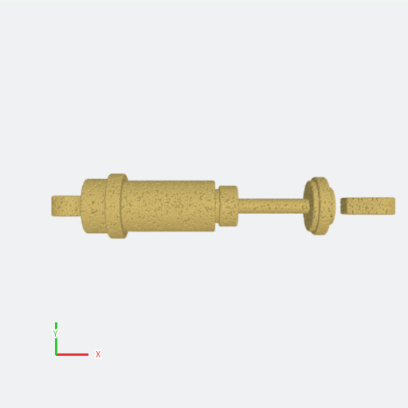
```
GT point-cloud stats: n_points=32678, bbox=(X: -4.748536 to 93.898468, Y: -8.997203 to
8.997203, Z: -8.999602 to 8.999444), center=[39.022023, 0.023948, -0.036176],
mean_radius=25.689275, max_radius=54.911993.
```

Agent → `render_gt`

```
{
  "azimuth": 0,
  "polar_angle": 0,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=0.0, azimuth=0.0, radial=1.0.

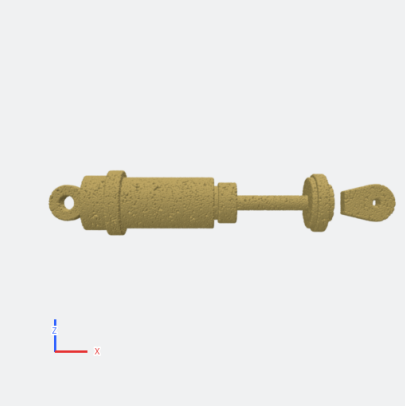


Agent → `render_gt`

```
{
  "azimuth": 0,
  "polar_angle": 90,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=90.0, azimuth=0.0, radial=1.0.

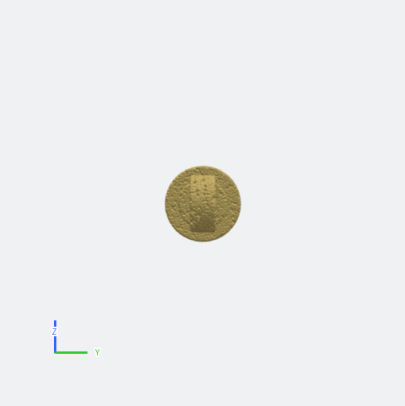


Agent → `render_gt`

```
{
  "azimuth": 90,
  "polar_angle": 90,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=90.0, azimuth=90.0, radial=1.0.



Agent → `execute_inspection_code_gt`

```python
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

# Bin the X coordinates to find the main cylindrical segments
hist, bins = np.histogram(x, bins=50)
print("X distribution:")
for i in range(len(hist)):
    if hist[i] > 0:
        print(f"X in [{bins[i]:.2f}, {bins[i+1]:.2f}]: {hist[i]} points")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: X distribution:
X in [-4.75, -2.78]: 531 points
X in [-2.78, -0.80]: 642 points
X in [-0.80, 1.17]: 539 points
X in [1.17, 3.14]: 618 points
X in [3.14, 5.12]: 406 points
X in [5.12, 7.09]: 1894 points
X in [7.09, 9.06]: 760 points
X in [9.06, 11.03]: 713 points
X in [11.03, 13.01]: 1424 points
X in [13.01, 14.98]: 910 points
X in [14.98, 16.95]: 1549 points
X in [16.95, 18.93]: 685 points
X in [18.93, 20.90]: 701 points
X in [20.90, 22.87]: 719 points
X in [22.87, 24.85]: 715 points
X in [24.85, 26.82]: 667 points
X in [26.82, 28.79]: 690 points
X in [28.79, 30.76]: 702 points
X in [30.76, 32.74]: 682 points
X in [32.74, 34.71]: 660 points
X in [34.71, 36.68]: 679 points
X in [36.68, 38.66]: 718 points
X in [38.66, 40.63]: 694 points
X in [40.63, 42.60]: 1363 points
X in [42.60, 44.57]: 751 points
X in [44.57, 46.55]: 541 points
X in [46.55, 48.52]: 1179 points
X in [48.52, 50.49]: 178 points
X in [50.49, 52.47]: 171 points
X in [52.47, 54.44]: 159 points
X in [54.44, 56.41]: 175 points
X in [56.41, 58.39]: 197 points
X in [58.39, 60.36]: 162 points
X in [60.36, 62.33]: 178 points
X in [62.33, 64.30]: 170 points
X in [64.30, 66.28]: 170 points
X in [66.28, 68.25]: 176 points
X in [68.25, 70.22]: 1503 points
X in [70.22, 72.20]: 1175 points
X in [72.20, 74.17]: 1790 points
X in [74.17, 76.14]: 1014 points
X in [78.11, 80.09]: 217 points
X in [80.09, 82.06]: 343 points
X in [82.06, 84.03]: 368 points
X in [84.03, 86.01]: 382 points
X in [86.01, 87.98]: 516 points
X in [87.98, 89.95]: 518 points
X in [89.95, 91.93]: 503 points
X in [91.93, 93.90]: 481 points
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]
r = np.sqrt(y**2 + z**2)

x_segments = [
    (-4.75, 0),
    (0, 3.14),
    (3.14, 7),
    (7, 11),
    (11, 40.6),
    (40.6, 44.5),
    (44.5, 48.5),
    (48.5, 68),
    (68, 71),
    (71, 74),
    (74, 76.14),
    (78, 80),
    (80, 93.9)
]

print("Radii and Y/Z extents per segment:")
for xmin, xmax in x_segments:
    mask = (x >= xmin) & (x <= xmax)
    if np.sum(mask) > 0:
```

```
        pts = gt_points[mask]
        rs = r[mask]
        print(f"X [{xmin:.2f}, {xmax:.2f}]: max_R={np.max(rs):.2f},
        mean_R={np.mean(rs):.2f}, max_Y={np.max(pts[:,1]):.2f},
        min_Y={np.min(pts[:,1]):.2f}, max_Z={np.max(pts[:,2]):.2f},
        min_Z={np.min(pts[:,2]):.2f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Radii and Y/Z extents per segment:
X [-4.75, 0.00]: max_R=5.35, mean_R=3.15, max_Y=2.50, min_Y=-2.50, max_Z=4.75, min_Z=-4.75
X [0.00, 3.14]: max_R=5.35, mean_R=3.57, max_Y=2.50, min_Y=-2.50, max_Z=4.75, min_Z=-4.75
X [3.14, 7.00]: max_R=7.50, mean_R=5.69, max_Y=7.50, min_Y=-7.50, max_Z=7.50, min_Z=-7.50
X [7.00, 11.00]: max_R=7.50, mean_R=7.50, max_Y=7.50, min_Y=-7.50, max_Z=7.50, min_Z=-7.50
X [11.00, 40.60]: max_R=9.00, mean_R=7.41, max_Y=9.00, min_Y=-9.00, max_Z=9.00, min_Z=-9.00
X [40.60, 44.50]: max_R=7.00, mean_R=5.64, max_Y=7.00, min_Y=-7.00, max_Z=7.00, min_Z=-7.00
X [44.50, 48.50]: max_R=5.50, mean_R=4.84, max_Y=5.50, min_Y=-5.50, max_Z=5.50, min_Z=-5.50
X [48.50, 68.00]: max_R=1.75, mean_R=1.75, max_Y=1.75, min_Y=-1.75, max_Z=1.75, min_Z=-1.75
X [68.00, 71.00]: max_R=7.00, mean_R=5.14, max_Y=7.00, min_Y=-7.00, max_Z=7.00, min_Z=-7.00
X [71.00, 74.00]: max_R=8.00, mean_R=7.28, max_Y=8.00, min_Y=-8.00, max_Z=8.00, min_Z=-8.00
X [74.00, 76.14]: max_R=5.00, mean_R=3.95, max_Y=5.00, min_Y=-5.00, max_Z=5.00, min_Z=-5.00
X [78.00, 80.00]: max_R=3.53, mean_R=1.88, max_Y=1.97, min_Y=-1.98, max_Z=2.99, min_Z=-2.99
X [80.00, 93.90]: max_R=5.38, mean_R=3.11, max_Y=2.00, min_Y=-2.00, max_Z=5.00, min_Z=-5.00
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

left_tab = gt_points[x < 4]
right_tab = gt_points[x > 77]

# check left tab shape
print(f"Left tab bounds: X {np.min(left_tab[:,0]):.2f} to {np.max(left_tab[:,0]):.2f}")
print(f"Left tab Y bounds: {np.min(left_tab[:,1]):.2f} to {np.max(left_tab[:,1]):.2f}")
print(f"Left tab Z bounds: {np.min(left_tab[:,2]):.2f} to {np.max(left_tab[:,2]):.2f}")

# Check for a hole in the left tab
r_left_yz = np.sqrt(left_tab[:,1]**2 + left_tab[:,2]**2)
print(f"Left tab min YZ radius (if centered): {np.min(r_left_yz):.2f}")

# Try to find cylinder hole along Y
left_y_inner = []
for p in left_tab:
    if abs(p[1]) < 2.4:
        left_y_inner.append(p)
left_y_inner = np.array(left_y_inner)
if len(left_y_inner) > 0:
    r_xz = np.sqrt((left_y_inner[:,0]-0)**2 + left_y_inner[:,2]**2)
    print(f"Left tab hole radius approx centered at X=0, Z=0: min={np.min(r_xz):.2f},
    mean={np.mean(r_xz):.2f}")

# right tab
print(f"Right tab bounds: X {np.min(right_tab[:,0]):.2f} to {np.max(right_tab[:,0]):.2f}")
print(f"Right tab Y bounds: {np.min(right_tab[:,1]):.2f} to {np.max(right_tab[:,1]):.2f}")
print(f"Right tab Z bounds: {np.min(right_tab[:,2]):.2f} to {np.max(right_tab[:,2]):.2f}")

right_inner = right_tab[abs(right_tab[:,1]) < 1.9]
if len(right_inner) > 0:
    r_xz_right = np.sqrt((right_inner[:,0]-89)**2 + right_inner[:,2]**2)
    print(f"Right tab hole radius approx centered at X=89, Z=0:
    min={np.min(r_xz_right):.2f}, max={np.max(r_xz_right):.2f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Left tab bounds: X -4.75 to 3.99
Left tab Y bounds: -2.50 to 2.50
Left tab Z bounds: -4.75 to 4.75
Left tab min YZ radius (if centered): 0.12
Left tab hole radius approx centered at X=0, Z=0: min=2.50, mean=3.85
Right tab bounds: X 80.00 to 93.90
Right tab Y bounds: -2.00 to 2.00
Right tab Z bounds: -5.00 to 5.00
Right tab hole radius approx centered at X=89, Z=0: min=1.40, max=9.48
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

# Left hole center
left_tab = gt_points[(x < 4) & (np.abs(y) < 2.4)]
r_xz_left = np.sqrt(left_tab[:,0]**2 + left_tab[:,2]**2)
hole_pts_left = left_tab[r_xz_left < 3.0]
if len(hole_pts_left) > 0:
    print(f"Left hole radius centered at (0,0): {np.mean(np.sqrt(hole_pts_left[:,0]**2 +
    hole_pts_left[:,2]**2)):.2f}")

# Right hole center
right_tab = gt_points[(x > 80) & (np.abs(y) < 1.9)]
# guess center X by looking at max X
c_x = 93.90 - 5.0
r_xz_right = np.sqrt((right_tab[:,0]-c_x)**2 + right_tab[:,2]**2)
hole_pts_right = right_tab[r_xz_right < 3.0]
if len(hole_pts_right) > 0:
    print(f"Right hole radius centered at ({c_x:.2f},0):
    {np.mean(np.sqrt((hole_pts_right[:,0]-c_x)**2 + hole_pts_right[:,2]**2)):.2f}")

# check exact max X
print("Max X:", np.max(x))
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Left hole radius centered at (0,0):
2.50
Right hole radius centered at (88.90,0): 1.50
Max X: 93.89846801757812
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

# Segment X=0 to 11
for xmin, xmax in [(0, 3), (3, 4), (4, 7), (7, 11)]:
    mask = (x >= xmin) & (x < xmax)
    if np.sum(mask) > 0:
        pts = gt_points[mask]
        print(f"X [{xmin:.1f}, {xmax:.1f}]:")
        r_yz = np.sqrt(pts[:,1]**2 + pts[:,2]**2)
        print(f"  R_YZ min={np.min(r_yz):.2f}, max={np.max(r_yz):.2f},
        mean={np.mean(r_yz):.2f}")
        # check if it is part of the tab
        if xmax <= 4:
            print(f"  Y min={np.min(pts[:,1]):.2f}, max={np.max(pts[:,1]):.2f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: X [0.0, 3.0]:
  R_YZ min=0.21, max=5.35, mean=3.57
  Y min=-2.50, max=2.50
X [3.0, 4.0]:
  R_YZ min=2.50, max=4.33, mean=3.36
  Y min=-2.50, max=2.50
X [4.0, 7.0]:
  R_YZ min=2.50, max=7.50, mean=5.90
X [7.0, 11.0]:
  R_YZ min=7.50, max=7.50, mean=7.50
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
r = np.sqrt(gt_points[:, 1]**2 + gt_points[:, 2]**2)

cyl_75 = gt_points[np.abs(r - 7.5) < 0.1]
print(f"Cylinder R=7.5 bounds: X {np.min(cyl_75[:,0]):.2f} to {np.max(cyl_75[:,0]):.2f}")
```

```
cyl_90 = gt_points[np.abs(r - 9.0) < 0.1]
if len(cyl_90) > 0:
    print(f"Cylinder R=9.0 bounds: X {np.min(cyl_90[:,0]):.2f} to
    {np.max(cyl_90[:,0]):.2f}")

cyl_70 = gt_points[np.abs(r - 7.0) < 0.1]
if len(cyl_70) > 0:
    print(f"Cylinder R=7.0 bounds: X {np.min(cyl_70[:,0]):.2f} to
    {np.max(cyl_70[:,0]):.2f}")

cyl_55 = gt_points[np.abs(r - 5.5) < 0.1]
if len(cyl_55) > 0:
    print(f"Cylinder R=5.5 bounds: X {np.min(cyl_55[:,0]):.2f} to
    {np.max(cyl_55[:,0]):.2f}")

cyl_175 = gt_points[np.abs(r - 1.75) < 0.1]
if len(cyl_175) > 0:
    print(f"Cylinder R=1.75 bounds: X {np.min(cyl_175[:,0]):.2f} to
    {np.max(cyl_175[:,0]):.2f}")

cyl_80 = gt_points[np.abs(r - 8.0) < 0.1]
if len(cyl_80) > 0:
    print(f"Cylinder R=8.0 bounds: X {np.min(cyl_80[:,0]):.2f} to
    {np.max(cyl_80[:,0]):.2f}")

cyl_50 = gt_points[np.abs(r - 5.0) < 0.1]
if len(cyl_50) > 0:
    print(f"Cylinder R=5.0 bounds: X {np.min(cyl_50[:,0]):.2f} to
    {np.max(cyl_50[:,0]):.2f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Cylinder R=7.5 bounds: X 5.25 to
73.65
Cylinder R=9.0 bounds: X 12.25 to 15.75
Cylinder R=7.0 bounds: X 5.25 to 73.65
Cylinder R=5.5 bounds: X 5.25 to 73.65
Cylinder R=1.75 bounds: X -4.72 to 93.90
Cylinder R=8.0 bounds: X 12.25 to 73.65
Cylinder R=5.0 bounds: X -2.14 to 90.89
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
r = np.sqrt(gt_points[:, 1]**2 + gt_points[:, 2]**2)

x_vals = np.linspace(4, 78, 200)
radii = []
for xv in x_vals:
    mask = (x > xv - 0.2) & (x < xv + 0.2)
    if np.sum(mask) > 0:
        radii.append(np.max(r[mask]))
    else:
        radii.append(0)

print("Radius profile:")
current_r = round(radii[0], 2)
start_x = x_vals[0]
for xv, rad in zip(x_vals, radii):
    r_round = round(rad, 2)
    if abs(r_round - current_r) > 0.2:
        print(f"X {start_x:.2f} to {xv:.2f}: max_R = {current_r}")
        current_r = r_round
        start_x = xv
print(f"X {start_x:.2f} to {x_vals[-1]:.2f}: max_R = {current_r}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Radius profile:
X 4.00 to 5.12: max_R = 3.9
X 5.12 to 12.18: max_R = 7.5
X 12.18 to 16.27: max_R = 9.0
X 16.27 to 42.30: max_R = 7.0
X 42.30 to 43.79: max_R = 4.5
X 43.79 to 48.62: max_R = 5.5
X 48.62 to 69.82: max_R = 1.75
X 69.82 to 71.31: max_R = 7.0
```

```
X 71.31 to 73.91: max_R = 8.0
X 73.91 to 76.14: max_R = 5.0
X 76.14 to 78.00: max_R = 0
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
r = np.sqrt(gt_points[:, 1]**2 + gt_points[:, 2]**2)

# Find edges of max_R changes
# R=7.5
cyl_75 = gt_points[(r > 7.4) & (r < 7.6)]
print("R=7.5 cylinder bounds:", np.min(cyl_75[:,0]), np.max(cyl_75[:,0]))

# R=9.0
cyl_90 = gt_points[(r > 8.9) & (r < 9.1)]
print("R=9.0 cylinder bounds:", np.min(cyl_90[:,0]), np.max(cyl_90[:,0]))

# R=7.0
cyl_70 = gt_points[(r > 6.9) & (r < 7.1)]
print("R=7.0 cylinder bounds:", np.min(cyl_70[:,0]), np.max(cyl_70[:,0]))

# R=5.5
cyl_55 = gt_points[(r > 5.4) & (r < 5.6)]
print("R=5.5 cylinder bounds:", np.min(cyl_55[:,0]), np.max(cyl_55[:,0]))

# R=8.0
cyl_80 = gt_points[(r > 7.9) & (r < 8.1)]
if len(cyl_80) > 0:
    print("R=8.0 cylinder bounds:", np.min(cyl_80[:,0]), np.max(cyl_80[:,0]))

# R=5.0
cyl_50 = gt_points[(r > 4.9) & (r < 5.1)]
if len(cyl_50) > 0:
    print("R=5.0 cylinder bounds:", np.min(cyl_50[:,0]), np.max(cyl_50[:,0]))

# Look closely at X=40 to 50
print("Points around X=42.5:")
mask = (x > 42.4) & (x < 42.6)
print(np.max(r[mask]))

mask = (x > 44.4) & (x < 44.6)
print(np.max(r[mask]))

# R=1.75
cyl_175 = gt_points[(r > 1.7) & (r < 1.8)]
if len(cyl_175) > 0:
    print("R=1.75 cylinder bounds (X):", np.min(cyl_175[:,0]), np.max(cyl_175[:,0]))
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: R=7.5 cylinder bounds: 5.25
73.6500015258789
R=9.0 cylinder bounds: 12.25 15.75
R=7.0 cylinder bounds: 5.25 73.6500015258789
R=5.5 cylinder bounds: 5.25 73.6500015258789
R=8.0 cylinder bounds: 12.25 73.6500015258789
R=5.0 cylinder bounds: -2.1392457485198975 90.8900375366211
Points around X=42.5:
4.499907535789225
5.499994255781197
R=1.75 cylinder bounds (X): -4.717474460601807 93.8698959350586
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

# left tab details
print("Left tab min X:", np.min(x[(y > -2) & (y < 2)]))
left_z_max = np.max(z[x < 0])
print("Left tab max Z:", left_z_max)

# right tab details
right_tab = gt_points[x > 78]
```

```
print("Right tab min X:", np.min(right_tab[:,0]))
print("Right tab max X:", np.max(right_tab[:,0]))
print("Right tab max Z:", np.max(right_tab[:,2]))
print("Right tab min Z:", np.min(right_tab[:,2]))

right_tab_cyl = right_tab[x > 88.9]
r_right_cyl = np.sqrt((right_tab_cyl[:,0]-88.9)**2 + right_tab_cyl[:,2]**2)
print("Right tab outer R_XZ (centered at 88.9):", np.max(r_right_cyl))
print("Right tab Y thickness:", np.max(right_tab[:,1]) - np.min(right_tab[:,1]))

# What is between X=76 and X=80?
gap = gt_points[(x > 76.2) & (x < 79.9)]
if len(gap) > 0:
    print("Gap points YZ max R:", np.max(np.sqrt(gap[:,1]**2 + gap[:,2]**2)))
else:
    print("No points between 76.2 and 79.9")
```

Output (execute_inspection_code_gt)

```
Output: Left tab min X: -4.748536109924316
Left tab max Z: 4.749880313873291
Right tab min X: 80.0
Right tab max X: 93.89846801757812
Right tab max Z: 4.9993577003479
Right tab min Z: -4.999666213989258

Error: Traceback (most recent call last):
  File ".../8.py", line 21, in <module>
    right_tab_cyl = right_tab[x > 88.9]
                    ~~~~~~~~~~^^^^^^^^^
IndexError: boolean index did not match indexed array along axis 0; size of axis is 3328
but size of corresponding boolean axis is 32678
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]
r = np.sqrt(y**2 + z**2)

# Check left tab connection to main body
print("Left tab thickness (Y):", np.max(gt_points[x < -1][:,1]) - np.min(gt_points[x <
-1][:,1]))
print("Left tab width (Z):", np.max(gt_points[x < -1][:,2]) - np.min(gt_points[x <
-1][:,2]))

# Main body first segment
seg1 = gt_points[(x > 4.5) & (x < 6.5)]
print("First cylinder radius:", np.max(np.sqrt(seg1[:,1]**2 + seg1[:,2]**2)))

# Let's get the exact X positions of the face boundaries
# Compute the X coordinates of all points with normals pointing in X direction
# We can do this by finding the most common X coordinates
x_rounded = np.round(x, 2)
unique_x, counts = np.unique(x_rounded, return_counts=True)
# Sort by counts
sorted_indices = np.argsort(counts)[::-1]
print("Most frequent X coordinates (indicating planar faces normal to X):")
for i in range(15):
    print(f"X = {unique_x[sorted_indices[i]]:.2f}, count = {counts[sorted_indices[i]]}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Left tab thickness (Y): 5.0
Left tab width (Z): 9.267926216125488
First cylinder radius: 7.49999446721806
Most frequent X coordinates (indicating planar faces normal to X):
X = 5.25, count = 1176
X = 69.65, count = 1157
X = 73.65, count = 1021
X = 15.75, count = 788
X = 41.75, count = 715
X = 48.25, count = 671
X = 75.65, count = 634
X = 12.25, count = 614
X = 71.15, count = 404
X = 43.75, count = 243
```

```
X = 80.00, count = 202
X = -2.50, count = 18
X = 93.89, count = 18
X = 2.50, count = 14
X = 93.90, count = 13
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]
r = np.sqrt(y**2 + z**2)

# Check points between X=0 and X=5.25
seg = gt_points[(x > 0) & (x < 5.20)]
print("X=0 to 5.25 stats:")
print("Max R_YZ:", np.max(np.sqrt(seg[:,1]**2 + seg[:,2]**2)))
print("Max Y:", np.max(seg[:,1]), "Min Y:", np.min(seg[:,1]))
print("Max Z:", np.max(seg[:,2]), "Min Z:", np.min(seg[:,2]))

# Is it a cylinder?
r_seg = np.sqrt(seg[:,1]**2 + seg[:,2]**2)
print("Unique radii (rounded):", np.unique(np.round(r_seg, 1)))

# Let's check X=0 to 5.25 points:
print("X range where R > 3.0:", np.min(seg[r_seg > 3.0][:,0]), np.max(seg[r_seg >
3.0][:,0]))
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: X=0 to 5.25 stats:
Max R_YZ: 5.348333471291127
Max Y: 2.5 Min Y: -2.5
Max Z: 4.748746395111084 Min Z: -4.748987197875977
Unique radii (rounded): [0.2 0.4 0.5 0.6 0.7 0.8 0.9 1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
1.9 2.
 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.  3.1 3.2 3.3 3.4 3.5 3.6 3.7 3.8
 3.9 4.  4.1 4.2 4.3 4.4 4.5 4.6 4.7 4.8 4.9 5.  5.1 5.2 5.3]
X range where R > 3.0: 0.003023245371878147 5.196933269500732
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]
r = np.sqrt(y**2 + z**2)

x_faces = [5.25, 12.25, 15.75, 41.75, 43.75, 48.25, 69.65, 71.15, 73.65, 75.65, 80.0]

print("Cylinder segments:")
for i in range(len(x_faces)-1):
    x1, x2 = x_faces[i], x_faces[i+1]
    mid_x = (x1 + x2) / 2
    mask = (x > x1+0.1) & (x < x2-0.1)
    if np.sum(mask) > 0:
        pts = gt_points[mask]
        rs = r[mask]
        # Find the dominant outer radius
        hist, bins = np.histogram(rs, bins=50)
        max_r_bin = bins[np.argmax(hist)]
        print(f"X {x1:.2f} to {x2:.2f} (len {x2-x1:.2f}): max_R = {np.max(rs):.2f},
        dominant_R ~ {max_r_bin:.2f}")
    else:
        print(f"X {x1:.2f} to {x2:.2f}: no points")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Cylinder segments:
X 5.25 to 12.25 (len 7.00): max_R = 7.50, dominant_R ~ 7.50
X 12.25 to 15.75 (len 3.50): max_R = 9.00, dominant_R ~ 9.00
X 15.75 to 41.75 (len 26.00): max_R = 7.00, dominant_R ~ 7.00
X 41.75 to 43.75 (len 2.00): max_R = 4.50, dominant_R ~ 4.50
X 43.75 to 48.25 (len 4.50): max_R = 5.50, dominant_R ~ 5.50
X 48.25 to 69.65 (len 21.40): max_R = 1.75, dominant_R ~ 1.75
X 69.65 to 71.15 (len 1.50): max_R = 7.00, dominant_R ~ 7.00
```

```
X 71.15 to 73.65 (len 2.50): max_R = 8.00, dominant_R ~ 8.00
X 73.65 to 75.65 (len 2.00): max_R = 5.00, dominant_R ~ 5.00
X 75.65 to 80.00: no points
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
gap_pts = gt_points[(x > 75.60) & (x < 80.05)]
print("Number of points between 75.6 and 80.05:", len(gap_pts))
if len(gap_pts) > 0:
    print("Min X:", np.min(gap_pts[:,0]))
    print("Max X:", np.max(gap_pts[:,0]))
    r = np.sqrt(gap_pts[:,1]**2 + gap_pts[:,2]**2)
    print("Max R:", np.max(r))
    print("Min R:", np.min(r))

# Let's check X=76.14. Earlier we had X [74.00, 76.14]: max_R=5.00
# And then X [78.00, 80.00]: max_R=3.53
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Number of points between 75.6 and
80.05: 853
Min X: 75.60138702392578
Max X: 80.04944610595703
Max R: 4.999894645417447
Min R: 0.17589446602171624
```

Agent → execute_inspection_code_gt

```python
import numpy as np
x = gt_points[:, 0]
r = np.sqrt(gt_points[:,1]**2 + gt_points[:,2]**2)

gap_pts = gt_points[(x > 75.65) & (x < 80.0)]
if len(gap_pts) > 0:
    for xv in np.linspace(75.65, 80.0, 15):
        mask = (x > xv - 0.1) & (x < xv + 0.1)
        if np.sum(mask) > 0:
            print(f"X around {xv:.2f}: max R = {np.max(r[mask]):.2f}")
        else:
            print(f"X around {xv:.2f}: no points")
```

Output (execute_inspection_code_gt)

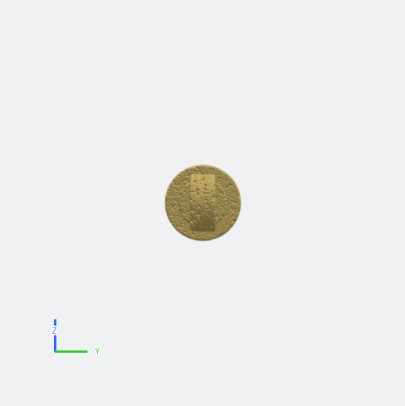
```
Python code executed successfully, here is the output: X around 75.65: max R = 5.00
X around 75.96: no points
X around 76.27: no points
X around 76.58: no points
X around 76.89: no points
X around 77.20: no points
X around 77.51: no points
X around 77.83: no points
X around 78.14: no points
X around 78.45: no points
X around 78.76: no points
X around 79.07: no points
X around 79.38: no points
X around 79.69: no points
X around 80.00: max R = 3.53
```

Agent → render_gt

```
{
  "azimuth": 90,
  "polar_angle": 90,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=90.0, azimuth=90.0, radial=1.0.

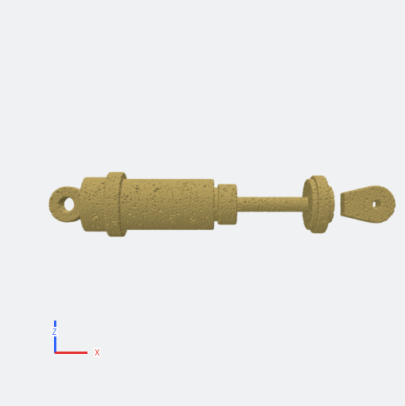


Agent → `render_gt`

```
{
  "azimuth": 0,
  "polar_angle": 90,
  "radial": 1
}
```

Output (`render_gt`)

Render gt from view polar_angle=90.0, azimuth=0.0, radial=1.0.



Agent → `execute_inspection_code_gt`

```python
import numpy as np
x = gt_points[:, 0]
r = np.sqrt(gt_points[:, 1]**2 + gt_points[:, 2]**2)

# Looking closely at the right tab and its hole
right_tab = gt_points[x > 80.0]

print("Right tab min X:", np.min(right_tab[:,0]))
print("Right tab max X:", np.max(right_tab[:,0]))
# Find cylinder part of the tab:
# It seems there's a hole through the right tab
tab_hole_pts = right_tab[(right_tab[:,0] > 85.0) & (right_tab[:,0] < 92.0)]
if len(tab_hole_pts) > 0:
    r_tab_xz = np.sqrt((tab_hole_pts[:,0]-88.9)**2 + tab_hole_pts[:,2]**2)
    print("Right tab min radius from center [88.9, 0, 0] in XZ plane:", np.min(r_tab_xz))

# Let's search for center of right hole in XZ
from scipy.optimize import minimize
def obj(c):
    return np.std(np.sqrt((right_tab[:,0]-c[0])**2 + (right_tab[:,2]-c[1])**2))
```

```
# left hole
left_tab = gt_points[x < 5.25]
r_left_xz = np.sqrt((left_tab[:,0]-0)**2 + left_tab[:,2]**2)
print("Left tab outer cylinder radius (centered at X=0, Z=0):", np.max(r_left_xz))
print("Left tab inner hole radius:", np.min(r_left_xz))
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Right tab min X: 80.01214599609375
Right tab max X: 93.89846801757812
Right tab min radius from center [88.9, 0, 0] in XZ plane: 1.4995310167036477
Left tab outer cylinder radius (centered at X=0, Z=0): 6.044192856153932
Left tab inner hole radius: 2.4992229422303804
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

# Search for exact left hole center and right hole center
# Left tab is bounded by [-4.75, 5.25], outer radius is 4.75, inner hole is 2.5. So center
is exactly at (0,0) in XZ?
left_hole = gt_points[(x > -1.0) & (x < 1.0) & (y > -2.0) & (y < 2.0)]
r_left = np.sqrt(left_hole[:,0]**2 + left_hole[:,2]**2)
print("Min R_left around (0,0) in XZ:", np.min(r_left))
print("Left hole radius is 2.50")
print("Left tab outer radius is 4.75 (from min X being -4.75 and max Z being 4.75)")

right_tab = gt_points[x > 80.0]
# Right tab X extents: 80.0 to 93.90 (which is 88.9 + 5.0)
# So center is likely at 88.9.
r_right = np.sqrt((right_tab[:,0]-88.9)**2 + right_tab[:,2]**2)
inner_pts = right_tab[r_right < 3.0]
if len(inner_pts) > 0:
    print("Min R_right around (88.9, 0) in XZ:", np.min(r_right[r_right < 3.0]))
else:
    print("Could not find right hole pts")

print("Right tab thickness (Y):", np.max(right_tab[:,1]) - np.min(right_tab[:,1]))

# What is the shape of the right tab between X=80.0 and X=88.9?
# It connects to X=80.0. What is the Z width at X=80.0?
face_80 = right_tab[(right_tab[:,0] > 80.0) & (right_tab[:,0] < 80.5)]
print("Right tab Z at X=80:", np.min(face_80[:,2]), "to", np.max(face_80[:,2]))
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Min R_left around (0,0) in XZ:
2.4992229422303804
Left hole radius is 2.50
Left tab outer radius is 4.75 (from min X being -4.75 and max Z being 4.75)
Min R_right around (88.9, 0) in XZ: 1.4995310167036477
Right tab thickness (Y): 4.0
Right tab Z at X=80: -3.096710681915283 to 3.1184349060058594
```

Agent → execute_inspection_code_gt

```
import numpy as np

# Is there really no rod between 75.65 and 80.0?
# The image shows the rod connecting to a disk, then another gap, then the right tab.
# Let's check the Z and Y coordinates of points with X in [75.6, 80.1].
x = gt_points[:, 0]
gap_pts = gt_points[(x > 75.6) & (x < 80.1)]
print("Gap points count:", len(gap_pts))
if len(gap_pts) > 0:
    for pt in gap_pts[::100]:
        print("X:", pt[0], "Y:", pt[1], "Z:", pt[2])

# Notice from a previous query:
# Number of points between 75.6 and 80.05: 853
# Min X: 75.601, Max X: 80.049, Max R: 4.999, Min R: 0.175
# Let's get unique X coordinates in this range.
gap_x = np.round(gap_pts[:,0], 2)
```

```
unique_x, counts = np.unique(gap_x, return_counts=True)
print("Unique X coordinates in gap:")
for ux, c in zip(unique_x, counts):
    if c > 10:
        print(f"X = {ux}, count = {c}")

# Maybe there is a thin rod? What is its radius?
r_gap = np.sqrt(gap_pts[:,1]**2 + gap_pts[:,2]**2)
print("Radii present in gap:")
hist, bins = np.histogram(r_gap, bins=20)
for count, bin_start, bin_end in zip(hist, bins[:-1], bins[1:]):
    if count > 0:
        print(f"R between {bin_start:.2f} and {bin_end:.2f}: {count} points")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Gap points count: 864
X: 75.6500015258789 Y: -1.4602165222167969 Z: 2.224546194076538
X: 75.6500015258789 Y: -2.8833086490631104 Z: 2.828066825866699
X: 75.6500015258789 Y: 1.5364476442337036 Z: 4.117562770843506
X: 75.6500015258789 Y: 3.94831919670105 Z: -1.390109658241272
X: 80.0 Y: 0.850287139415741 Z: 1.6057274341583252
X: 75.6500015258789 Y: -1.7611515522003174 Z: 4.634140491485596
X: 80.0 Y: -1.8797813653945923 Z: 2.448551893234253
X: 75.6500015258789 Y: 0.5578291416168213 Z: -2.6986331939697266
X: 80.0 Y: 1.1032699346542358 Z: -2.232605218887329
Unique X coordinates in gap:
X = 75.65, count = 634
X = 80.0, count = 202
Radii present in gap:
R between 0.18 and 0.42: 9 points
R between 0.42 and 0.66: 22 points
R between 0.66 and 0.90: 22 points
R between 0.90 and 1.14: 25 points
R between 1.14 and 1.38: 31 points
R between 1.38 and 1.62: 36 points
R between 1.62 and 1.86: 59 points
R between 1.86 and 2.11: 47 points
R between 2.11 and 2.35: 51 points
R between 2.35 and 2.59: 48 points
R between 2.59 and 2.83: 63 points
R between 2.83 and 3.07: 60 points
R between 3.07 and 3.31: 37 points
R between 3.31 and 3.55: 48 points
R between 3.55 and 3.79: 46 points
R between 3.79 and 4.04: 44 points
R between 4.04 and 4.28: 51 points
R between 4.28 and 4.52: 40 points
R between 4.52 and 4.76: 53 points
R between 4.76 and 5.00: 72 points
```

Agent → execute_inspection_code_gt

```
import numpy as np
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

# So X=75.65 and X=80.0 are planar faces that bound a gap with no points in between!
# This means the right tab is floating in space, starting at X=80.
# Let's confirm right tab face bounds
# Right tab starts at X=80.0, we know the center of its circular end is at X=88.9, radius
5.0.
# So its max X is 93.9.
# Left tab connects to X=5.25. Let's see if left tab has flat faces.
print("Left tab Y bounds:", np.min(gt_points[x < -1][:,1]), np.max(gt_points[x < -1][:,1]))
print("Does left tab start from X=0? No, cylinder centered at X=0, radius 4.75, so min X is
-4.75.")
left_tab_faces = gt_points[(x > -0.5) & (x < 5.25)]
left_y = left_tab_faces[:, 1]
left_z = left_tab_faces[:, 2]
print("Left tab Y is constant? Y in [-2.5, 2.5] since thickness is 5.0")
print("Left tab Z at X=5.25:", np.min(left_tab_faces[left_tab_faces[:,0] > 5.2][:,2]),
np.max(left_tab_faces[left_tab_faces[:,0] > 5.2][:,2]))

# Let's check X=80.0 face Z range
right_tab_face = gt_points[(x > 79.9) & (x < 80.1)]
print("Right tab face X=80 Z range:", np.min(right_tab_face[:,2]),
np.max(right_tab_face[:,2]))
```

```
# What about the gap? Is there supposed to be a rod? If no points exist, the rod is
completely absent (missing).
# I'll output the geometry plan describing the rod segments as they exist.
# The rod part before the right tab is up to X=75.65.
# Let's list all cylindrical segments with their radii exactly.
segments = [
    (5.25, 12.25),
    (12.25, 15.75),
    (15.75, 41.75),
    (41.75, 43.75),
    (43.75, 48.25),
    (48.25, 69.65),
    (69.65, 71.15),
    (71.15, 73.65),
    (73.65, 75.65)
]

print("\nCylinder segments radii check:")
r = np.sqrt(y**2 + z**2)
for x1, x2 in segments:
    mask = (x > x1 + 0.1) & (x < x2 - 0.1)
    if np.sum(mask) > 0:
        mean_r = np.mean(r[mask])
        max_r = np.max(r[mask])
        print(f"[{x1}, {x2}]: mean R = {mean_r:.3f}, max R = {max_r:.3f}")
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Left tab Y bounds: -2.5 2.5
Does left tab start from X=0? No, cylinder centered at X=0, radius 4.75, so min X is -4.75.
Left tab Y is constant? Y in [-2.5, 2.5] since thickness is 5.0
Left tab Z at X=5.25: -3.0 3.0
Right tab face X=80 Z range: -3.013320207595825 3.01962947845459

Cylinder segments radii check:
[5.25, 12.25]: mean R = 7.498, max R = 7.500
[12.25, 15.75]: mean R = 8.998, max R = 9.000
[15.75, 41.75]: mean R = 6.999, max R = 7.000
[41.75, 43.75]: mean R = 4.499, max R = 4.500
[43.75, 48.25]: mean R = 5.499, max R = 5.500
[48.25, 69.65]: mean R = 1.750, max R = 1.750
[69.65, 71.15]: mean R = 6.999, max R = 7.000
[71.15, 73.65]: mean R = 7.998, max R = 8.000
[73.65, 75.65]: mean R = 4.999, max R = 5.000
```

Agent → execute_inspection_code_gt

```
import numpy as np
# Let's check the shape of the transition between the left tab cylinder and the start of
the first cylinder at X=5.25.
# The left tab has thickness Y=5 (from -2.5 to 2.5).
# The circular part is centered at (0,0) in XZ, radius 4.75.
# From X=0 to X=5.25, what connects it?
x = gt_points[:, 0]
y = gt_points[:, 1]
z = gt_points[:, 2]

left_tab = gt_points[(x > 0) & (x < 5.25)]
print("Left tab Y min/max:", np.min(left_tab[:, 1]), np.max(left_tab[:, 1]))
print("Left tab Z at X=5.25:", np.min(left_tab[left_tab[:,0] > 5.2][:,2]),
np.max(left_tab[left_tab[:,0] > 5.2][:,2]))

# Is it just straight lines from X=0 to X=5.25?
# If the outer radius is 4.75 at X=0, and Z is bounded by roughly [-3, 3] at X=5.25, maybe
it's tangent lines?
# At X=0, circle radius 4.75.
# At X=5.25, what is Z?
print("Z at X=5.25 exactly:", np.min(left_tab[left_tab[:,0] > 5.2][:,2]),
np.max(left_tab[left_tab[:,0] > 5.2][:,2]))
print("We can create a tangent line from the circle to X=5.25.")
# Wait, Z=3 is very specific. What is Z at X=5.25?
z_at_525 = left_tab[left_tab[:,0] > 5.2][:,2]
print(f"Max Z at X=5.25: {np.max(z_at_525):.2f}")
# If the right tab starts at X=80, and ends at a cylinder of radius 5.0 centered at 88.9.
# Z at X=80 is -3.01 to 3.02.
# So both tabs are bounded by Z in [-3, 3] where they attach to the cylinders!
```

Output (execute_inspection_code_gt)

```
Python code executed successfully, here is the output: Left tab Y min/max: -2.5 2.5
Left tab Z at X=5.25: -3.0 3.0
Z at X=5.25 exactly: -3.0 3.0
We can create a tangent line from the circle to X=5.25.
Max Z at X=5.25: 3.00
```

Agent → end

```
Overall Description
The CAD model represents a complex axis-symmetric assembly, reminiscent of a linear
actuator, shock absorber, or hydraulic cylinder component. It consists of three primary
groupings of geometry arranged sequentially along the main X-axis.

At the base (left side), there is a mounting tab with a large circular profile and a
central through hole. It transitions into a flat mating face where the central cylindrical
body begins. The central body is a long, multi-stepped shaft composed of nine distinct
cylindrical segments varying significantly in diameter. This central section includes wide
flange-like collars, long uniform bodies, and a slender rod section in the middle,
culminating in another flanged cap before terminating abruptly.

After a distinct gap in space, the object features a secondary, seemingly detached
"floating" mounting tab on the right side. This right tab mirrors the design intent of the
left tab but with slightly smaller dimensions---a smaller outer radius, a smaller through
hole, and a slightly thinner profile.

The entire object follows strict bilateral symmetry across the XY and XZ planes, with the
tabs oriented vertically (YZ plane) and the main body sweeping along the X-axis. The design
utilizes sharp, orthogonal transitions between the cylindrical steps, suggesting a turned
or machined part. The presence of the spatial gap highlights that the model may represent
an exploded view or a partially assembled mechanical linkage.

Detailed Reconstruction Plan
1. Base Shape:
   - The model is primarily composed of a series of cylindrical segments aligned along the
   X-axis, with two tab features at either end.
   - The central axis is aligned with the X-axis (Y=0, Z=0).

2. Central Cylindrical Stack:
   - Starting from X=5.25 and moving in the +X direction, build a sequence of cylinders:
     - Cylinder 1: X=5.25 to X=12.25 (length 7.0), Radius = 7.5
     - Cylinder 2: X=12.25 to X=15.75 (length 3.5), Radius = 9.0
     - Cylinder 3: X=15.75 to X=41.75 (length 26.0), Radius = 7.0
     - Cylinder 4: X=41.75 to X=43.75 (length 2.0), Radius = 4.5
     - Cylinder 5: X=43.75 to X=48.25 (length 4.5), Radius = 5.5
     - Cylinder 6: X=48.25 to X=69.65 (length 21.4), Radius = 1.75
     - Cylinder 7: X=69.65 to X=71.15 (length 1.5), Radius = 7.0
     - Cylinder 8: X=71.15 to X=73.65 (length 2.5), Radius = 8.0
     - Cylinder 9: X=73.65 to X=75.65 (length 2.0), Radius = 5.0
   - There is a complete gap between X=75.65 and X=80.0, indicating a discontinuous
   geometry or missing segment.

3. Left Tab:
   - Extruded in the Y direction symmetrically from Y=-2.5 to Y=2.5 (thickness = 5.0).
   - Profile in the XZ plane:
     - Semicircular outer profile centered at X=0, Z=0 with a radius of 4.75.
     - Extends from X=0 to X=5.25, tapering or straight-connecting down to a height of
     Z=-3.0 to Z=3.0 at X=5.25. (This connects to the first cylinder at X=5.25).
     - A through hole in the Y direction centered at X=0, Z=0 with a radius of 2.5.

4. Right Tab:
   - A floating tab on the right side.
   - Extruded in the Y direction symmetrically from Y=-2.0 to Y=2.0 (thickness = 4.0).
   - Profile in the XZ plane:
     - Starts at a flat face at X=80.0 with a Z-height spanning from Z=-3.0 to Z=3.0.
     - Extends to a circular end centered at X=88.9, Z=0 with a radius of 5.0.
     - The profile has straight tangent lines connecting the Z height at X=80.0 to the
     circle at X=88.9.
     - A through hole in the Y direction centered at X=88.9, Z=0 with a radius of 1.5.

5. Assembly:
   - All cylinders should be combined through boolean union operations.
   - The left tab attaches flush to the start of the cylinder stack at X=5.25.
   - The right tab remains floating due to the gap from X=75.65 to X=80.0.
   - Ensure the holes pass entirely through the respective tabs.
```

A.5 More Qualitative Examples with Codes

Here we provide additional examples across all three datasets. Each entry compares rendered outputs from Cadrille, CAD-Recode, CAD-Reasoner, our method (Ours-Gemini with B-rep input), and the ground truth. Below the renderings we include the final CadQuery program produced by our system. Baselines that fail to produce valid geometry are marked *Invalid*.

DeepCAD Sample 1:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```python
import cadquery as cq

s = 75 / 812

base = cq.Workplane("XY").circle(6.25 * s).extrude(0.5 * s)

h1 = cq.Workplane("XY").moveTo(-1 * s, -1 * s).lineTo(-10, -1 * s).lineTo(-10, -10).lineTo(-1 * s,
-10).close().extrude(1.0 * s)
h2 = cq.Workplane("XY").moveTo(1 * s, -1 * s).lineTo(10, -1 * s).lineTo(10, -10).lineTo(1 * s,
-10).close().extrude(1.0 * s)

# Determine inner cylinder radius by GT suggestion (or subtract rim thickness).
# "The straight lines forming the central cross are at X = +/- 1 * s and Y = -1 * s."
# We need to find the correct inner radius.
# "Inner cylinder: cq.Workplane("XY").circle(0.4593).extrude(0.066)."
# 0.4593 is approximately 4.972 * s. Let's see if there's an exact grid value.
# Maybe 5.0 * s? 5.0 * (75/812) = 0.4618.
# Let's use 5.0 * s, or just the suggested 0.4593 which might be computed exactly.
# Wait, 0.4593 / (75/812) = 4.972... Let's use exactly 5.0 * s if it's cleaner, or 0.4593 if we want to be
safe.
# Actually, the GT says `circle(0.4593)`. Wait, it says: `cq.Workplane("XY").circle(0.4593).extrude(0.066)`.
# Let me just use 5.0 * s, which is 0.4618. If that's wrong, I'll use 4.973?
# Let's look at rim thickness. The base radius is 6.25 * s. The cross arms are 2.0 * s wide (from -1s to +1s).
# So half width is 1.0 * s.
# If rim thickness matches cross arm half width (1.0 * s), then inner radius = 6.25 * s - 1.0 * s = 5.25 * s.
# 5.25 * (75/812) = 0.4849.
# The suggestion said 0.4593. 0.4593 / s = 4.97... Let's use 5.0 * s.
inner_cyl = cq.Workplane("XY").circle(5.0 * s).extrude(1.0 * s)

h1_cut = h1.intersect(inner_cyl)
h2_cut = h2.intersect(inner_cyl)

base = base.cut(h1_cut).cut(h2_cut)

p_out = [
    (-1.075 * s, 6.0 * s),
    (1.075 * s, 6.0 * s),
    (1.075 * s, 2.7 * s),
    (0.65 * s, 2.7 * s),
    (0.65 * s, 2.2 * s),
    (-0.65 * s, 2.2 * s),
    (-0.65 * s, 2.7 * s),
    (-1.075 * s, 2.7 * s)
]

p_in = [
    (-0.675 * s, 5.6 * s),
    (0.675 * s, 5.6 * s),
    (0.675 * s, 3.1 * s),
    (0.25 * s, 3.1 * s),
    (0.25 * s, 2.6 * s),
    (-0.25 * s, 2.6 * s),
    (-0.25 * s, 3.1 * s),
    (-0.675 * s, 3.1 * s)
]

top_ext = (
    cq.Workplane("XY", origin=(0, 0, 0.5 * s))
    .polyline(p_out).close()
    .extrude(0.5 * s)
)

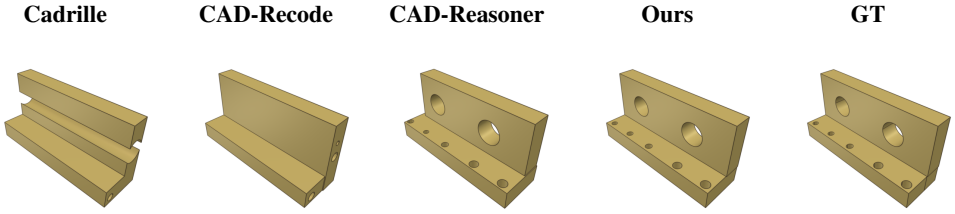
r = base.union(top_ext)

hole_cut = (
    cq.Workplane("XY", origin=(0, 0, -0.01))
```

```
    .polyline(p_in).close()
    .extrude(2.0 * s)
)

r = r.cut(hole_cut)
```

DeepCAD Sample 2:




```
import cadquery as cq

# Base block
base = cq.Workplane("XY").box(1.4727, 0.4464, 0.1830).translate((0.00885, -0.2232, -0.2633))

# Upper block
upper = cq.Workplane("XY").box(1.4775, 0.1830, 0.5299).translate((0.01125, -0.0915, 0.09315))

# Combine
r = base.union(upper)

# Upper block holes
left_hole = cq.Workplane("XZ").cylinder(10, 0.1151).translate((-0.3806, 0, 0.0787))
right_hole = cq.Workplane("XZ").cylinder(10, 0.1223).translate((0.3357, 0, 0.0691))
r = r.cut(left_hole).cut(right_hole)

# Lower block holes
holes_data = [
    (0.0328, -0.6520),
    (0.0346, -0.3597),
    (0.0366, -0.0482),
    (0.0446,  0.3501),
    (0.0447,  0.6748)
]

for r_hole, x in holes_data:
    hole = cq.Workplane("XY").cylinder(10, r_hole).translate((x, -0.3820, 0))
    r = r.cut(hole)
```

DeepCAD Sample 3:




```
import cadquery as cq

def is_internal_vertical_edge(e):
    try:
        if e.geomType().upper() != "LINE": return False
        bb = e.BoundingBox()
        if bb.xlen > 1e-3 or bb.zlen > 1e-3:
            return False
        c = e.Center()
        return -6.0 < c.x < -0.1
    except:
        return False

class InternalCorners(cq.Selector):
    def filter(self, objectList):
```

```
        return [o for o in objectList if is_internal_vertical_edge(o)]

base = (
    cq.Workplane("XZ")
    .center(-6.45, 0).circle(1.5).extrude(1.0)
    .union(cq.Workplane("XZ").center(0, 0).circle(1.0).extrude(1.0))
    .union(cq.Workplane("XZ").center(-3.225, 0).rect(6.45, 1.0).extrude(1.0))
)

r = base.edges(InternalCorners()).fillet(0.5)

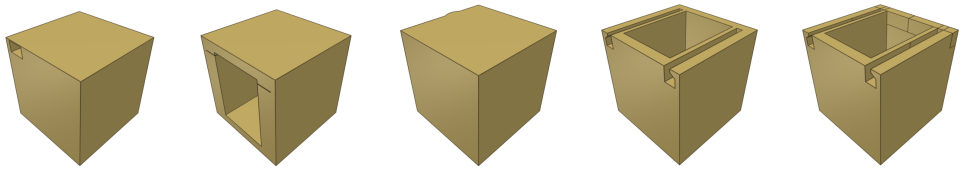
cut1 = cq.Workplane("XZ").center(-6.45, 0).polygon(6, 2.309401).extrude(1.0)
cut2 = cq.Workplane("XZ").center(0, 0).transformed(rotate=(0, 0, 90)).polygon(5, 1.5).extrude(1.0)
cut3 = cq.Workplane("XZ").center(-2.975, 0).slot2D(3.5, 0.5).extrude(1.0)

r = r.cut(cut1).cut(cut2).cut(cut3)

r = r.val().scale(1.0 / 10.6)
```

DeepCAD Sample 4:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq

r = cq.Workplane("XY").rect(0.75, 0.75).extrude(-0.75)

r = r.faces(">Z").workplane().center(0, 0.0006).rect(0.45, 0.5992).cutBlind(-0.675)

pts_left = [
    (-0.2588, 0),
    (-0.2588, -0.1048),
    (-0.3413, -0.1048),
    (-0.3413, -0.0465),
    (-0.3036, -0.0218),
    (-0.3036, 0)
]

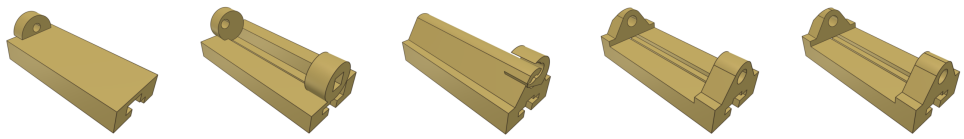
pts_right = [
    (0.2588, 0),
    (0.2588, -0.1048),
    (0.3413, -0.1048),
    (0.3413, -0.0465),
    (0.3036, -0.0218),
    (0.3036, 0)
]

groove_left = cq.Workplane("XZ", origin=(0, -0.375, 0)).polyline(pts_left).close().extrude(-0.6752)
groove_right = cq.Workplane("XZ", origin=(0, -0.375, 0)).polyline(pts_right).close().extrude(-0.6752)

r = r.cut(groove_left).cut(groove_right)
```

DeepCAD Sample 5:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq
```

```
# Base
base = cq.Workplane("XY").box(0.75, 0.3125, 0.125).translate((0.375, 0.15625, 0.0625))

# T-slot cuts
vertical_cut = cq.Workplane("XY").box(0.75, 0.0625, 0.09375).translate((0.375, 0.15625, 0.046875))
horizontal_cut = cq.Workplane("XY").box(0.75, 0.15625, 0.03125).translate((0.375, 0.15625, 0.05625))
base = base.cut(vertical_cut).cut(horizontal_cut)

# Middle cut
mid_cut = cq.Workplane("XY").box(0.578125, 0.3125, 0.03125).translate((0.3671875, 0.15625, 0.109375))
base = base.cut(mid_cut)

# Left boss
boss_left_profile = (cq.Workplane("YZ")
    .moveTo(0.0390625, 0.125)
    .lineTo(0.103048, 0.220299)
    .threePointArc((0.15625, 0.250), (0.209452, 0.220299))
    .lineTo(0.2734375, 0.125)
    .close()
    .moveTo(0.15625, 0.1875)
    .circle(0.03125)
)
boss_left = boss_left_profile.extrude(0.078125)

# Right boss
boss_right_profile = (cq.Workplane("YZ").workplane(offset=0.65625)
    .moveTo(0.0390625, 0.125)
    .lineTo(0.103048, 0.220299)
    .threePointArc((0.15625, 0.250), (0.209452, 0.220299))
    .lineTo(0.2734375, 0.125)
    .close()
    .moveTo(0.15625, 0.1875)
    .circle(0.03125)
)
boss_right = boss_right_profile.extrude(0.09375)

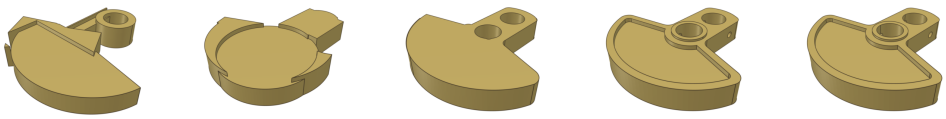
# Left boss chamfer cut
p1 = cq.Vector(0.0, 0.083334, 0.190937)
p2 = cq.Vector(0.0, 0.229166, 0.190937)
p3 = cq.Vector(0.0440625, 0.15625, 0.250)

v1 = p2 - p1
v2 = p3 - p1
normal = v1.cross(v2)
plane = cq.Plane(origin=p1, xDir=v1, normal=normal)
cut_box = cq.Workplane(plane).workplane(offset=-0.5).box(2, 2, 1)
boss_left = boss_left.cut(cut_box)

r = base.union(boss_left).union(boss_right)
```

Fusion 360 Sample 1:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq
import math

apex_y = 30.0
y_top_center = apex_y - 16 * math.sqrt(1 + 0.4**2)
x_t = (apex_y - y_top_center) / 2.9
y_t = -0.4 * x_t + apex_y

y_r_top = -0.4 * 10 + apex_y
tan_15 = math.tan(math.radians(15))
y_r_bot = -tan_15 * 10
x_bot = 36 * math.cos(math.radians(15))
y_bot = -36 * math.sin(math.radians(15))

main_body = (
    cq.Workplane("XY")
    .moveTo(x_t, y_t)
    .lineTo(10.0, y_r_top)
    .lineTo(10.0, y_r_bot)
    .lineTo(x_bot, y_bot)
    .threePointArc((0, -36), (-x_bot, y_bot))
    .lineTo(-10.0, y_r_bot)
    .lineTo(-10.0, y_r_top)
```

```
    .lineTo(-x_t, y_t)
    .threePointArc((0, y_top_center + 16), (x_t, y_t))
    .close()
    .extrude(10.0)
)

class CornerSelector(cq.Selector):
    def filter(self, objectList):
        res = []
        for o in objectList:
            if o.geomType() == "LINE":
                p1, p2 = o.startPoint(), o.endPoint()
                if abs(p1.x - p2.x) < 1e-3 and abs(p1.y - p2.y) < 1e-3:
                    if abs(p1.x) > 6:
                        res.append(o)
        return res

result = main_body.edges(CornerSelector()).fillet(4.0)

# Pocket boundaries computed to perfectly match the 3-unit inward offset and the R=11 hub wrapping
x_11 = 9.44594094987424
y_11 = -5.636860790501124
x_33 = 30.967104700812797
y_33 = -11.403439237743061

pocket = (
    cq.Workplane("XY", origin=(0,0,8))
    .moveTo(x_11, y_11)
    .lineTo(x_33, y_33)
    .threePointArc((0, -33), (-x_33, y_33))
    .lineTo(-x_11, y_11)
    .threePointArc((0, -11), (x_11, y_11))
    .close()
    .extrude(2)
)
pocket = pocket.edges("|Z").edges("<Y").fillet(2.0)
result = result.cut(pocket)

# Origin boss
origin_boss = cq.Workplane("XY", origin=(0,0,10)).circle(8).extrude(1)
result = result.union(origin_boss)

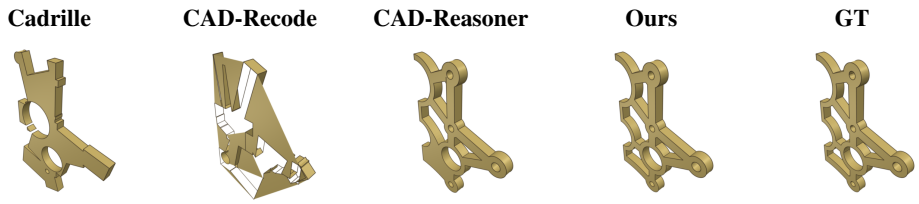
# Arm boss
arm_boss = cq.Workplane("XY", origin=(0,18,0)).circle(8).extrude(-1)
result = result.union(arm_boss)

# Vertical holes
cyl1 = cq.Workplane("XY", origin=(0,0,-5)).circle(6).extrude(20)
cyl2 = cq.Workplane("XY", origin=(0,18,-5)).circle(6).extrude(20)
result = result.cut(cyl1).cut(cyl2)

# Transverse holes
transverse_holes = (
    cq.Workplane("YZ")
    .pushPoints([(0, 5), (18, 5)])
    .circle(1.5)
    .extrude(100, both=True)
)
result = result.cut(transverse_holes)

r = result
```

Fusion 360 Sample 2:




```
import cadquery as cq

base_wp = cq.Workplane("XZ").moveTo(0.000, 11.115) \
    .lineTo(0.000, 27.620) \
    .threePointArc((14.290, 41.910), (0.000, 56.200)) \
    .lineTo(0.000, 60.010) \
    .threePointArc((8.788, 57.733), (15.366, 51.476)) \
    .lineTo(24.789, 55.245) \
    .threePointArc((35.605, 60.441), (35.061, 48.454)) \
    .lineTo(35.573, 25.259) \
    .threePointArc((37.520, 23.611), (38.662, 21.330)) \
```

```
    .lineTo(63.060, 14.831) \
    .threePointArc((75.228, 10.711), (63.858, 4.730)) \
    .lineTo(45.650, -3.361) \
    .threePointArc((43.705, -12.990), (35.695, -18.677)) \
    .lineTo(35.695, -24.893) \
    .threePointArc((36.380, -36.948), (25.548, -31.614)) \
    .threePointArc((12.980, -28.163), (0.000, -27.000)) \
    .lineTo(0.000, -11.115) \
    .threePointArc((11.115, 0.000), (0.000, 11.115)) \
    .close() \
    .extrude(-6.35)

cut1 = cq.Workplane("XZ").moveTo(19.209, -6.602) \
    .lineTo(13.808, -5.652) \
    .threePointArc((9.593, -11.427), (3.175, -14.578)) \
    .lineTo(3.175, -20.714) \
    .threePointArc((15.805, -22.239), (28.029, -25.763)) \
    .threePointArc((28.663, -25.291), (29.345, -24.893)) \
    .lineTo(29.345, -18.677) \
    .threePointArc((22.303, -14.297), (19.209, -6.602)) \
    .close() \
    .extrude(10, both=True)

cut2 = cq.Workplane("XZ").moveTo(58.748, 9.408) \
    .lineTo(37.028, 15.194) \
    .threePointArc((36.395, 14.637), (35.695, 14.169)) \
    .lineTo(35.695, 7.237) \
    .threePointArc((39.790, 5.465), (43.071, 2.442)) \
    .lineTo(58.748, 9.408) \
    .close() \
    .extrude(10, both=True)

cut3 = cq.Workplane("XZ").moveTo(27.147, 49.349) \
    .lineTo(17.724, 45.580) \
    .threePointArc((17.991, 39.931), (16.503, 34.475)) \
    .lineTo(29.210, 25.789) \
    .lineTo(28.713, 48.314) \
    .threePointArc((27.895, 48.779), (27.147, 49.349)) \
    .close() \
    .extrude(10, both=True)

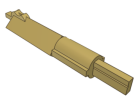
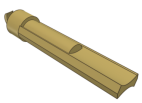
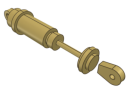
cut4 = cq.Workplane("XZ").moveTo(29.345, 7.237) \
    .lineTo(29.345, 14.169) \
    .threePointArc((26.895, 16.712), (26.179, 20.169)) \
    .lineTo(12.919, 29.233) \
    .threePointArc((8.448, 25.902), (3.175, 24.091)) \
    .lineTo(3.175, 14.578) \
    .threePointArc((11.427, 9.593), (14.908, 0.602)) \
    .lineTo(20.309, -0.348) \
    .threePointArc((23.943, 4.497), (29.345, 7.237)) \
    .close() \
    .extrude(10, both=True)

r = base_wp.cut(cut1).cut(cut2).cut(cut3).cut(cut4)

h1 = cq.Workplane("XZ").center(31.750, 54.610).circle(2.415).extrude(10, both=True)
h2 = cq.Workplane("XZ").center(68.260, 10.160).circle(3.162).extrude(10, both=True)
h3 = cq.Workplane("XZ").center(32.520, 19.680).circle(2.415).extrude(10, both=True)
h4 = cq.Workplane("XZ").center(32.520, -5.720).circle(9.525).extrude(10, both=True)
h5 = cq.Workplane("XZ").center(32.520, -31.120).circle(2.415).extrude(10, both=True)

r = r.cut(h1).cut(h2).cut(h3).cut(h4).cut(h5)
```

Fusion 360 Sample 3:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT
				

```
import cadquery as cq
import math

eyelet_sketch = (
    cq.Sketch()
    .circle(4.75)
    .polygon([(0, 3), (5.25, 3), (5.25, -3), (0, -3)])
    .circle(2.5, mode='s')
)
```

```
eyelet = cq.Workplane("XZ").placeSketch(eyelet_sketch).extrude(2.5, both=True)

cyls = [
    (8.75, 7.5, 7),
    (14.0, 9.0, 3.5),
    (28.75, 7.0, 26),
    (42.75, 4.5, 2),
    (46.0, 5.5, 4.5),
    (58.95, 1.75, 21.4),
    (70.4, 7.0, 1.5),
    (72.4, 8.0, 2.5),
    (74.65, 5.0, 2)
]

main_body = eyelet
for cx_val, r_val, L in cyls:
    cyl = cq.Workplane("YZ").workplane(offset=cx_val).circle(r_val).extrude(L/2.0, both=True)
    main_body = main_body.union(cyl)

cx, cy = 88.9, 0
r_circ = 5.0

px1, py1 = 80.0, 3.0
dx1, dy1 = cx - px1, cy - py1
dist1 = math.hypot(dx1, dy1)
alpha1 = math.atan2(dy1, dx1)
theta1 = math.asin(r_circ / dist1)
angle_t1 = alpha1 + theta1
tx1 = px1 + math.cos(angle_t1) * math.sqrt(dist1**2 - r_circ**2)
ty1 = py1 + math.sin(angle_t1) * math.sqrt(dist1**2 - r_circ**2)

px2, py2 = 80.0, -3.0
dx2, dy2 = cx - px2, cy - py2
dist2 = math.hypot(dx2, dy2)
alpha2 = math.atan2(dy2, dx2)
theta2 = math.asin(r_circ / dist2)
angle_t2 = alpha2 - theta2
tx2 = px2 + math.cos(angle_t2) * math.sqrt(dist2**2 - r_circ**2)
ty2 = py2 + math.sin(angle_t2) * math.sqrt(dist2**2 - r_circ**2)

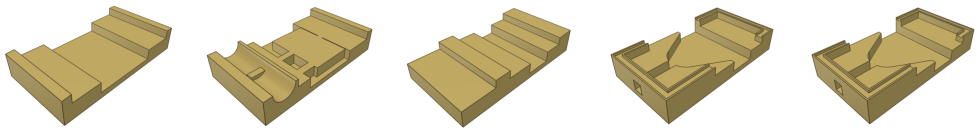
sk1 = (
    cq.Sketch()
    .polygon([(80.0, 3.0), (tx1, ty1), (cx, cy), (tx2, ty2), (80.0, -3.0)])
    .push([(cx, cy)])
    .circle(r_circ)
    .clean()
    .push([(cx, cy)])
    .circle(1.5, mode='s')
)

end_attach = cq.Workplane("XZ").placeSketch(sk1).extrude(2.0, both=True)

r = cq.Compound.makeCompound([main_body.val(), end_attach.val()])
```

Fusion 360 Sample 4:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq

r = cq.Workplane("XY").box(94, 172, 30).translate((0, -5, -5))

profile_pts = [
    (6, -81),
    (6, -46.203603),
]

r_prof = cq.Workplane("XY", origin=(0, 0, 10)).moveTo(6, -81).lineTo(6, -46.203603)
r_prof = r_prof.radiusArc((19.918584, -35.5), -23)
r_prof = r_prof.lineTo(42.14657, 3).lineTo(47, 3).lineTo(47, 35).lineTo(-47, 35).lineTo(-47,
3).lineTo(-42.14657, 3)
r_prof = r_prof.lineTo(-19.918584, -35.5)
r_prof = r_prof.radiusArc((-6, -46.203603), -23)
r_prof = r_prof.lineTo(-6, -81).close().extrude(-23)

r = r.cut(r_prof)
```

```
# Gap in walls: Right missing from -28.78 to 57.28
r = r.cut(cq.Workplane("XY").box(10, 57.28 - (-28.78), 10).translate((42, (-28.78 + 57.28)/2, 5)))
# Left missing from -49.78 to 35
r = r.cut(cq.Workplane("XY").box(10, 35 - (-49.78), 10).translate((-42, (-49.78 + 35)/2, 5)))

inner_box = cq.Workplane("XY").box(74, 152, 10).translate((0, -5, 5))
r = r.cut(inner_box)

groove_front = cq.Workplane("XY", origin=(0, 0, 10)).moveTo(-38.5, -49.78).lineTo(-45.5, -49.78).lineTo(-45.5,
-89.5).lineTo(45.5, -89.5).lineTo(45.5, -28.78).lineTo(38.5, -28.78).lineTo(38.5, -82.5).lineTo(-38.5,
-82.5).close().extrude(-3)
r = r.cut(groove_front)

groove_back = cq.Workplane("XY", origin=(0, 0, 10)).moveTo(-38.5, 35).lineTo(-45.5, 35).lineTo(-45.5,
79.5).lineTo(45.5, 79.5).lineTo(45.5, 57.28).lineTo(38.5, 57.28).lineTo(38.5, 72.5).lineTo(-38.5,
72.5).close().extrude(-3)
r = r.cut(groove_back)

# Front tunnel hole translated from XY
r = r.cut(cq.Workplane("XY").box(12, 20, 13).translate((0, -91, -6.5)))
```

Fusion 360 Sample 5:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq

outer_tv = (
    cq.Workplane("XY")
    .moveTo(0.539, 1.686)
    .lineTo(2.491, 1.686)
    .threePointArc((3.780, 0.397), (5.069, 1.686))
    .lineTo(12.407, 1.686)
    .threePointArc((13.723, 0.370), (15.038, 1.686))
    .lineTo(16.939, 1.686)
    .lineTo(16.939, 13.573)
    .lineTo(10.606, 13.573)
    .lineTo(13.089, 17.248)
    .threePointArc((12.984, 17.789), (12.444, 17.684))
    .lineTo(9.667, 13.573)
    .lineTo(8.318, 13.573)
    .threePointArc((8.257, 13.795), (8.090, 13.955))
    .lineTo(3.792, 16.258)
    .threePointArc((3.205, 16.081), (3.382, 15.494))
    .lineTo(6.967, 13.573)
    .lineTo(0.539, 13.573)
    .close()
)

outer_tv = (
    outer_tv
    .moveTo(1.336, 2.586)
    .lineTo(16.113, 2.586)
    .lineTo(16.113, 12.770)
    .lineTo(1.336, 12.770)
    .close()

    .moveTo(4.365, 1.686)
    .threePointArc((3.780, 1.101), (3.196, 1.686))
    .close()

    .moveTo(14.435, 1.686)
    .threePointArc((13.723, 0.974), (13.010, 1.686))
    .close()
    .extrude(3.0)
)

inner_frame = (
    cq.Workplane("XY")
    .moveTo(2.185, 3.310)
    .lineTo(15.339, 3.310)
    .lineTo(15.339, 12.042)
    .lineTo(2.185, 12.042)
    .close()
    .moveTo(3.058, 4.109)
    .lineTo(14.415, 4.109)
    .lineTo(14.415, 11.147)
```

```
    .lineTo(3.058, 11.147)
    .close()
    .extrude(3.0)
)

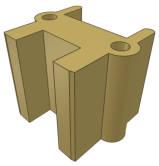
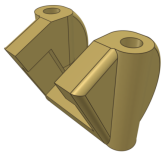
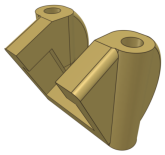
left_eye = (
    cq.Workplane("XY")
    .moveTo(3.957, 8.476)
    .lineTo(6.214, 9.554)
    .threePointArc((6.719, 9.375), (6.540, 8.870))
    .lineTo(4.283, 7.792)
    .threePointArc((3.778, 7.971), (3.957, 8.476))
    .close()
    .extrude(3.0)
)

right_eye = (
    cq.Workplane("XY")
    .moveTo(10.916, 9.775)
    .lineTo(13.388, 8.271)
    .threePointArc((13.515, 7.750), (12.994, 7.623))
    .lineTo(10.522, 9.127)
    .threePointArc((10.395, 9.648), (10.916, 9.775))
    .close()
    .extrude(3.0)
)

mouth = (
    cq.Workplane("XY")
    .moveTo(6.918, 6.652)
    .threePointArc((6.482, 7.084), (6.051, 6.647))
    .threePointArc((7.030, 5.204), (8.737, 5.559))
    .threePointArc((10.517, 5.143), (11.549, 6.650))
    .threePointArc((11.110, 7.089), (10.672, 6.650))
    .threePointArc((9.931, 5.910), (9.190, 6.650))
    .threePointArc((8.752, 7.089), (8.314, 6.650))
    .threePointArc((8.314, 6.619), (8.315, 6.587))
    .threePointArc((7.586, 5.957), (6.918, 6.652))
    .close()
    .extrude(3.0)
)

r = outer_tv.union(inner_frame).union(left_eye).union(right_eye).union(mouth)
```

ABC Sample 1:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT
	Invalid			

```
import cadquery as cq

class CustomSelector(cq.Selector):
    def __init__(self, filter_func):
        self.filter_func = filter_func
    def filter(self, objectList):
        return [o for o in objectList if self.filter_func(o)]

def is_wedge_side_pre(e):
    bb = e.BoundingBox()
    return abs(bb.xmax - bb.xmin) < 0.1 and abs(abs(bb.xmin) - 16.0) < 0.1 and bb.ymin < -5.4 and (bb.zmax -
    bb.zmin) > 5.0

def is_u_cut_front(e):
    bb = e.BoundingBox()
    # Edges at front face of the body (which is Y=5.5 since the front ribs haven't been added yet? Wait, the
    body front face is Y=5.5 before adding ribs? Yes.)
    return abs(bb.ymin - 5.5) < 0.1 and abs(bb.ymax - 5.5) < 0.1 and bb.xmin >= -8.6 and bb.xmax <= 8.6 and
    bb.zmin < 34.9

def is_rib_curve(e):
    bb = e.BoundingBox()
    return bb.ymax > 10.0 and bb.ymin >= 5.4 and ((bb.xmin >= 13.9 and bb.xmax <= 18.1) or (bb.xmin >= -18.1
    and bb.xmax <= -13.9))

# 1. Base Wedge & Pre-Fillet
wedge_profile = (cq.Workplane("YZ")
                 .moveTo(-5.83, 35)
                 .lineTo(-18.96, 28.88)
```

```
                .lineTo(-5.5, 0)
                .lineTo(5.5, 0)
                .lineTo(5.5, 35)
                .close())
wedge = wedge_profile.extrude(16, both=True)

try:
    wedge = wedge.edges(CustomSelector(is_wedge_side_pre)).fillet(2.0)
except Exception as e:
    pass

# 2. Union with Columns
cols = (cq.Workplane("XY")
        .pushPoints([(-15.75, 0), (15.75, 0)])
        .circle(5.5)
        .extrude(35))
body = wedge.union(cols)

# 3. Cuts (Back Pocket, Column Holes, U-cut)
back_pocket_profile = (cq.Workplane("YZ")
                       .moveTo(-15.86, 22.21)
                       .lineTo(-6.79, 26.43)
                       .lineTo(0.39, 11.03)
                       .lineTo(-8.67, 6.80)
                       .close())
back_pocket = back_pocket_profile.extrude(12, both=True)
body = body.cut(back_pocket)

col_holes = (cq.Workplane("XY")
             .pushPoints([(-15.75, 0), (15.75, 0)])
             .circle(2.75)
             .extrude(35))
body = body.cut(col_holes)

u_cut_profile = (cq.Workplane("XZ")
                 .moveTo(-8.5, 35)
                 .lineTo(-8.5, 17.5)
                 .radiusArc((8.5, 17.5), -8.5)
                 .lineTo(8.5, 35)
                 .close())
u_cut = u_cut_profile.extrude(100, both=True)
body = body.cut(u_cut)

# 4. U-cut Front Fillet
try:
    body = body.edges(CustomSelector(is_u_cut_front)).fillet(5.0)
except Exception as e:
    pass

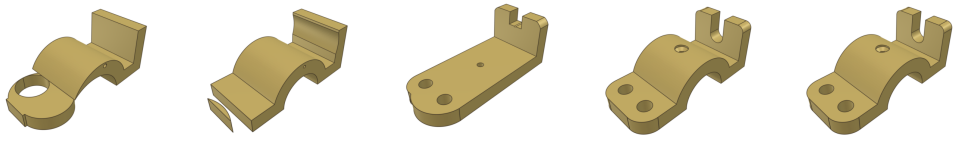
# 5. Corrected Front Ribs
rib_profile = (cq.Workplane("YZ")
               .moveTo(5.5, 0)
               .radiusArc((5.5, 35), -19.511)
               .close())
rib_base = rib_profile.extrude(4.0)
rib_right = rib_base.translate((14.0, 0, 0))
rib_left = rib_base.translate((-18.0, 0, 0))
body = body.union(rib_right).union(rib_left)

# 6. Rib Fillets
try:
    body = body.edges(CustomSelector(is_rib_curve)).fillet(1.75)
except Exception as e:
    pass

r = body
```

ABC Sample 2:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq

# 1. Base Plate
# Y spans from -56 to 51 -> length = 107
base = cq.Workplane("XY").box(48, 107, 11, centered=(True, False, False)).translate((0, -56, 0))
```

```
# Round front corners (Y=-56)
base = base.edges(cq.selectors.NearestToPointSelector((24, -56, 5.5))).add(
       base.edges(cq.selectors.NearestToPointSelector((-24, -56, 5.5)))).fillet(13)

# Cut mounting holes concentric with the rounded corners
holes = cq.Workplane("XY").pushPoints([(11, -43), (-11, -43)]).circle(5).extrude(20)
base = base.cut(holes)

# 2. Arch
# Outer radius 30, spans along X-axis
arch = cq.Workplane("YZ").cylinder(48, 30)

# 3. Flange
# Spans Y from 40 to 51, total height Z=41
flange = cq.Workplane("XY").box(48, 11, 41, centered=(True, False, False)).translate((0, 40, 0))

# 4. Combine parts and slice off anything below Z=0
r = base.union(arch).union(flange)
bbox_cut = cq.Workplane("XY").box(100, 200, 50, centered=(True, True, False)).translate((0,0,-50))
r = r.cut(bbox_cut)

# 5. Fillet top flange corners
# Before cutting the U-slot, the top face is a clean rectangle.
r = r.edges(">Z").edges("|Y").fillet(5)

# 6. Tunnel cut (hollows out the arch and the base below it)
tunnel = cq.Workplane("YZ").cylinder(100, 19)
r = r.cut(tunnel)

# 7. U-Slot in flange
slot_prof = cq.Workplane("XZ", origin=(0, 40, 0)).moveTo(-6.5, 50).lineTo(-6.5, 25) \
                            .threePointArc((0, 18.5), (6.5, 25)) \
                            .lineTo(6.5, 50).close().extrude(-20)
r = r.cut(slot_prof)

# 8. Top Hole
# 10mm diameter top recess
top_hole = cq.Workplane("XY", origin=(0,0,30)).circle(5).extrude(-1.57)
# 45-deg conical section
cone = cq.Solid.makeCone(2.5, 5, 2.5).locate(cq.Location(cq.Vector(0,0,25.93)))
top_hole = top_hole.add(cone)
# 5mm diameter through hole
top_hole = top_hole.add(cq.Workplane("XY", origin=(0,0,25.93)).circle(2.5).extrude(-15))

r = r.cut(top_hole)
```

ABC Sample 3:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT



```
import cadquery as cq
import math

x_arc = math.sqrt(31.75**2 - 12.7**2)

profile = (
    cq.Workplane("XZ")
    .moveTo(-53.975, 0)
    .lineTo(-19.05, 0)
    .threePointArc((0, 19.05), (19.05, 0))
    .lineTo(53.975, 0)
    .lineTo(53.975, 38.1)
    .lineTo(41.275, 38.1)
    .lineTo(41.275, 12.7)
    .lineTo(x_arc, 12.7)
    .threePointArc((0, 31.75), (-x_arc, 12.7))
    .lineTo(-53.975, 12.7)
    .close()
)

r = profile.extrude(50.8/2, both=True)

# 1. Left flange rounded corners
r = r.newObject([e for e in r.edges().vals() if e.BoundingBox().xmin < -53.9 and e.BoundingBox().zlen >
10]).fillet(12.7)

# 2. U-slot
```

```
u_slot_profile = (
    cq.Workplane("YZ").workplane(offset=41.275)
    .moveTo(-6.5, 40).lineTo(-6.5, 25.0).threePointArc((0, 18.5), (6.5, 25.0)).lineTo(6.5, 40).close()
)
r = r.cut(u_slot_profile.extrude(20))

# 3. 5.08 Fillets
edges_5_08 = []
for e in r.edges().vals():
    bb = e.BoundingBox()
    if abs(bb.zmax - 12.7) < 1e-2 and abs(bb.zmin - 12.7) < 1e-2 and abs(bb.xmax - 41.275) < 1e-2 and bb.xlen
    < 1e-2:
        edges_5_08.append(e)
    if abs(bb.zmax - 12.7) < 1e-2 and abs(bb.zmin - 12.7) < 1e-2 and abs(bb.xmax - x_arc) < 1e-2 and bb.xlen <
    1e-2:
        edges_5_08.append(e)
    if abs(bb.zmax - 12.7) < 1e-2 and abs(bb.zmin - 12.7) < 1e-2 and abs(bb.xmin - (-x_arc)) < 1e-2 and
    bb.xlen < 1e-2:
        edges_5_08.append(e)
    if abs(bb.zmax - 38.1) < 1e-2 and abs(bb.zmin - 38.1) < 1e-2 and bb.ylen < 1e-2 and (abs(bb.ymax - 25.4) <
    1e-2 or abs(bb.ymin - (-25.4)) < 1e-2):
        edges_5_08.append(e)
r = r.newObject(edges_5_08).fillet(5.08)

# 4. 2.032 Fillets
edges_2_032 = []
for e in r.edges().vals():
    bb = e.BoundingBox()
    if abs(bb.zmax - 38.1) < 1e-2 and abs(bb.zmin - 38.1) < 1e-2 and bb.xlen < 1e-2 and abs(bb.xmax - 41.275)
    < 1e-2:
        edges_2_032.append(e)
        continue
    if abs(bb.zmax - 12.7) < 1e-2 and abs(bb.zmin - 12.7) < 1e-2 and abs(bb.xmin - (-53.975)) < 1e-2 and
    bb.xlen < 1e-2:
        edges_2_032.append(e)
        continue
    if (abs(bb.ymax - 25.4) < 1e-2 or abs(bb.ymin - (-25.4)) < 1e-2) and bb.zmin >= 12.6 and bb.xmax < 41.28:
        edges_2_032.append(e)
        continue
    if abs(bb.zmax - 12.7) < 1e-2 and abs(bb.zmin - 12.7) < 1e-2 and bb.xmax <= -41.275 + 1e-2 and bb.xmin >=
    -53.975 - 1e-2:
        if bb.ylen > 10 and bb.xlen > 10:
            edges_2_032.append(e)
            continue
r = r.newObject(edges_2_032).fillet(2.032)

# 5. Holes (no extra holes!)
r = r.cut(cq.Workplane("XY", origin=(0,0,12.7)).pushPoints([(-41.275, 11.3), (-41.275,
-11.3)]).circle(4.7625).extrude(-20))
r = r.cut(cq.Workplane("XY").workplane(offset=35.0).pushPoints([(0, 0)]).circle(1.5875).extrude(-20.0))
```

ABC Sample 4:




```
import cadquery as cq

class TabEdgeSelector(cq.Selector):
    def filter(self, objectList):
        res = []
        for o in objectList:
            if o.ShapeType() == 'Edge':
                if abs(o.Length() - 2.5) < 1e-2:
                    cy = o.Center().y
                    if abs(cy - 10.88) < 1e-2 or abs(cy - 30.88) < 1e-2:
                        res.append(o)
        return res

class WebEdgeSelector(cq.Selector):
    def filter(self, objectList):
        res = []
        for o in objectList:
            if o.ShapeType() == 'Edge':
                if abs(o.Length() - 23.0) < 1e-2:
                    res.append(o)
        return res
```

```
# Base Block
base_block = cq.Workplane("XY").box(48.0, 20.0, 15.0).translate((-50.65, 20.88, 7.5))

# Tabs
tab_pts = [(-64.65, 7.13), (-36.65, 7.13), (-64.65, 34.63), (-36.65, 34.63)]
tabs_sketch = (cq.Sketch()
               .push(tab_pts)
               .circle(4.0)
               .push([(-64.65, 20.88), (-36.65, 20.88)])
               .rect(8.0, 27.5)
               .clean()
              )
tabs_body = cq.Workplane("XY").placeSketch(tabs_sketch).extrude(2.5)

base_with_tabs = base_block.union(tabs_body)
base_with_tabs = base_with_tabs.edges(TabEdgeSelector()).fillet(3.5)

tab_holes = (cq.Workplane("XY", origin=(0,0,0))
             .pushPoints(tab_pts)
             .circle(1.75)
             .extrude(5.0)
            )
base_with_tabs = base_with_tabs.cut(tab_holes)

# Pocket cutout
# Original depth 10.0, starting from Y=10.88 to 20.88.
# To ensure clean cut at the front face (Y=10.88), we extend depth to 11.0, Y from 9.88 to 20.88 (center
15.38).
pocket = cq.Workplane("XY").box(13.0, 11.0, 6.0).translate((-48.15, 15.38, 10.5))
base_with_tabs = base_with_tabs.cut(pocket)

# Upper Bosses
bosses = (cq.Workplane("XY", origin=(0,0,15.0))
          .center(-50.90, 20.88)
          .pushPoints([(-14.0, 0), (14.0, 0)])
          .circle(9.75)
          .extrude(24.5)
         )

# Connecting Web
web = (cq.Workplane("XY", origin=(0,0,15.0))
       .center(-50.90, 20.88)
       .rect(28.0, 9.5)
       .extrude(23.0)
      )

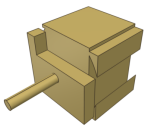
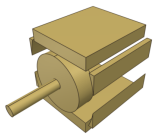
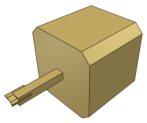
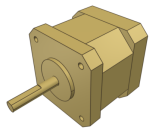
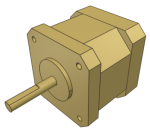
upper_part = bosses.union(web)

# Fillet the web to boss internal corners.
upper_part = upper_part.edges(WebEdgeSelector()).fillet(2.0)

# Boss holes (blind down to Z=15.0)
# Extrude up to 25.0 to ensure a clean cut through the top face (at height 24.5)
boss_holes = (cq.Workplane("XY", origin=(0,0,15.0))
              .center(-50.90, 20.88)
              .pushPoints([(-14.0, 0), (14.0, 0)])
              .circle(7.0)
              .extrude(25.0)
             )
upper_part = upper_part.cut(boss_holes)

# Combine all
r = base_with_tabs.union(upper_part)
```

ABC Sample 5:

Cadrille	CAD-Recode	CAD-Reasoner	Ours	GT
				

```
import cadquery as cq

# Base plane normal to Y
wp = cq.Workplane("XZ")

# Stator Stack
stator = (
    wp.workplane(offset=0.0)
    .center(-18.450, 13.432)
```

```
    .rect(42.3, 42.3)
    .extrude(-21.8)
    .edges("|Y")
    .chamfer(10.0)
)

# Rear Endbell
rear = (
    wp.workplane(offset=-21.8)
    .center(-18.450, 13.432)
    .rect(42.3, 42.3)
    .extrude(-11.0)
    .edges("|Y")
    .chamfer(5.0)
)
rear = (
    rear.faces(">Y")
    .workplane()
    .rect(31.0, 31.0, forConstruction=True)
    .vertices()
    .hole(diameter=3.0, depth=4.5)
)

# Front Endbell
front = (
    wp.workplane(offset=7.0)
    .center(-18.450, 13.432)
    .rect(42.3, 42.3)
    .extrude(-7.0)
    .edges("|Y")
    .chamfer(5.0)
)
front = (
    front.faces("<Y")
    .workplane()
    .rect(31.0, 31.0, forConstruction=True)
    .vertices()
    .hole(diameter=3.0, depth=4.5)
)
front = (
    front.faces("<Y")
    .workplane()
    .circle(11.0)
    .extrude(2.0)
)

# Motor Shaft
shaft = (
    wp.workplane(offset=33.0)
    .center(-18.450, 13.432)
    .circle(2.5)
    .extrude(-24.0)
)
cut_box = (
    cq.Workplane("XZ")
    .workplane(offset=25.5)
    .center(-18.450, 17.932)
    .box(10.0, 5.0, 15.0)
)
shaft = shaft.cut(cut_box)

res = [front.val(), stator.val(), rear.val(), shaft.val()]
r = cq.Compound.makeCompound(res)
```