CADFORGE: AGENTIC SINGLE-VIEW CAD RECONSTRUCTION WITH EXPLICIT GEOMETRY REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reconstructing editable parametric CAD models from a single-view image is of great practical value for modern manufacturing, yet remains challenging due to incomplete geometric observations and complex inter-part relationships. To address it, we propose ***CADForge***, an agentic framework that progressively converts a single image into CadQuery programs. CADForge decomposes an object into CAD-meaningful components and performs explicit geometric reasoning for each component, a process that first identifies CAD-relevant constraints and then translates them into precise modeling parameters through mathematical code. The inferred parameters then drive component-wise synthesis of executable CadQuery programs, with a review agent evaluating the resulting geometry and providing targeted feedback for iterative refinement. To further improve robustness and efficiency, CADForge incorporates a failure-guided toolkit construction mechanism to distill accumulated experience into tools, and maintains a compact parametric CAD memory for retrieving modeling context on demand. Experiments on diverse single- and multi-part objects show that CADForge consistently outperforms existing baselines in reconstruction fidelity and perceptual quality, demonstrating an effective approach to accurate single-view CAD reconstruction.

1 INTRODUCTION

Computer-aided design (CAD) plays an essential role in modern engineering (Narayan et al. (2013)), where 3D objects can be constructed through sequences of modeling operations and editable parameters. Compared with point clouds and meshes, parametric CAD models can provide precise boundary representations (B-Reps) and flexible editability, making them well suited for real-world applications such as furniture design (Scurtu et al. (2015)), and industrial manufacturing (Campi et al. (2022)). However, creating such models typically requires specialized expertise and considerable manual effort, driving growing interest in automatic parametric CAD generation.

Earlier work explored automatic CAD generation by learning neural representations of CAD construction sequences with dedicated network architectures (Wu et al. (2021); Xu et al. (2022; 2023)). With advances in large language models (LLMs) and vision-language models (VLMs), current work has made considerable progress in generating parametric CAD models from text descriptions (Khan et al. (2024b); Li et al. (2025a); Guan et al. (2025)), point clouds (Rukhovich et al. (2025); Kabisov et al. (2026); Li et al. (2026b)) or multi-view images (Kolodiazhnyi et al. (2026); Pyatov et al. (2026)), achieving strong performance in both geometric accuracy and structural validity.

However, accurate parametric CAD reconstruction from a single-view image, which offers significant practical value in industrial design and 3D printing (Sachs et al. (1990)), remains challenging. This task is inherently nontrivial as it requires a model not only to infer complete 3D geometry and structural relationships from partial visual observations but also to translate them into precise CAD modeling parameters and commands. Existing methods (Li et al. (2025c); Noeckel et al. (2025); Chen et al. (2025a); Doris et al. (2026a)) primarily rely on task-specific fine-tuning over curated image-CAD datasets and struggle to produce complex multi-part objects like furniture. When applied to in-the-wild images, they often produce distorted geometry, omit important components, or generate disconnected parts. More recently, some studies (Zhou et al. (2026); Lin et al. (2026)) have begun to explore agent-based 3D code generation. Although they can produce visually compelling

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.

results, they often incur substantial token cost and can't fully capture the underlying geometric constraints and inter-component spatial relationships, resulting in unfaithful reconstructions.

To address this challenge, we propose ***CADForge***, an agentic framework for single-view CAD reconstruction generation that converts an input image into a CadQuery program. Rather than generating the entire object monolithically, CADForge decomposes it into individual parts and constructs the model progressively. For each part, CADForge performs explicit geometry reasoning through two stages: *geometry observation* and *geometry programming*. In the first stage, a VLM extracts geometric patterns, proportional cues, and spatial relationships with previously generated parts from the input image. These observations are then translated into mathematical programs that leverage the current CAD state to solve for the key parameters required to construct the part. Based on the obtained specification and parameters, a code generator, assisted by a self-constructed toolkit, generates executable CadQuery code and integrates the new part into the existing model. We also introduce a reviewer agent to assess the result for geometric fidelity and topological validity, diagnose potential errors, and provide targeted feedback for iterative refinement. Throughout this process, CADForge introduces a compact parametric CAD memory to record the current modeling state, preserving cross-component consistency with minimal token overhead.

We construct an evaluation set covering both part-level and object-level CAD models collected from diverse datasets, and evaluate CADForge against existing methods on it. The results demonstrate that CADForge can more faithfully reconstruct the input image as a parametric CAD model while maintaining competitive token consumption. Our contributions can be summarized as:

- We propose CADForge, a multi-stage agentic framework tailored for single-view CAD reconstruction generation that decomposes an object into individual parts and progressively constructs a parametric CAD model through executable CadQuery code.
- To accurately capture geometric information from the input image, we introduce an explicit geometry reasoning mechanism that first infers relative geometric relationships among parts and then translates them into key modeling parameters. A compact parametric CAD memory is maintained to ground each reasoning step in the evolving construction state.
- We develop a comprehensive benchmark that covers both individual CAD parts and complete multi-part objects from diverse data sources. Experimental results demonstrate that CADForge achieves higher reconstruction fidelity than existing methods.

2 Related Work

Parametric CAD Generation. Learning-based CAD generation began with efforts to represent construction histories as sequences of sketches and geometric operations (Wu et al. (2021); Xu et al. (2022; 2023)). Building on these representations, advances in foundation models and large language models have inspired a growing body of work on conditional CAD generation. These methods translate textual descriptions into editable CAD sequences or executable programs, with some further incorporating explicit geometric controls or intermediate model states to support more complex construction procedures (Khan et al. (2024b); Li et al. (2024; 2025a); Zhang et al. (2025); Guan et al. (2025); Qi et al. (2026)). Alongside this development, another line of research models CAD generation as a reverse-engineering problem, seeking to recover the underlying construction process from geometric observations such as point clouds and multi-view images (Khan et al. (2024a); Liu et al. (2024); Ma et al. (2024); Rukhovich et al. (2025); Kabisov et al. (2026)). More recently, several methods have moved toward unified multimodal frameworks that accommodate heterogeneous inputs, including text, multi-view images, and point clouds, allowing different forms of design intent and geometric evidence to be handled within a shared generation backbone (Xu et al. (2024); Li et al. (2025b); Kolodiazhnyi et al. (2026); Pyatov et al. (2026)).

Compared with text- or geometry-conditioned settings, single-view CAD reconstruction still faces great challenges. Existing approaches broadly follow two directions: directly predicting CAD command sequences or executable programs (Alam & Ahmed (2024); Chen et al. (2025b;a); Doris et al. (2026a)), and recovering structured B-Reps using learned geometric priors and fitting (Li et al. (2025c)). Most of these methods rely on task-specific training with image-CAD datasets that largely consist of simple CAD models. Consequently, their performance often degrades on out-of-distribution images and relatively complicated objects. In contrast, CADForge adopts a training-free

agent-based framework, which leverages the visual, geometric, and programming priors of VLMs and generalizes more effectively without specialized fine-tuning.

Agentic 3D Modeling. Recent advances in large language models and vision-language models have spurred growing interest in agentic 3D modeling, where models plan modeling procedures, interact with 3D software, and refine their outputs through execution and visual feedback. Early systems primarily use LLMs to operate general-purpose 3D software such as Blender, translating natural-language instructions into coordinated modeling actions for object and scene creation (Hu et al. (2024); Sun et al. (2025); Zhou et al. (2025)). More work has gone beyond tool control by treating executable programs as structured representations of 3D objects themselves. Compared with direct mesh representations, code makes semantic parts and construction logic explicit, supports editing and reuse, and enables deterministic execution and automated verification (Lu et al. (2025); Dai et al. (2025)). Building on these advantages, some approaches generate compositional assets through specialized programmatic interfaces that explicitly represent parts, spatial relationships, and structural constraints (Barkley et al. (2026); Li et al. (2026a); Zhou et al. (2026); Lin et al. (2026)).However, these methods are mostly designed for text-conditioned generation. When applied to image-based reconstruction, they often struggle to recover key geometric patterns and inter-part relationships, and their outputs are not necessarily editable B-Reps. In contrast, CADForge is explicitly designed for single-view CAD reconstruction. It combines a tailored part-decomposition strategy with explicit geometric reasoning to recover geometric structure and spatial relationships from the input image and convert them into an editable parametric CAD model.

3 METHOD

In this section, we present CADForge, which reconstructs a CAD model from an arbitrary image. As illustrated in Fig. 1, CADForge first decomposes the complete object into a set of distinct yet interrelated components and then constructs them part by part (3.2). For each part, our geometry reasoning module carefully analyzes the input image and formulates mathematical programs to solve for the key geometric parameters (3.3). Assisted by a self-constructed toolkit, a code generator then synthesizes the corresponding CadQuery code (3.4). To ensure global consistency without introducing excessive context, we maintain a compact parametric CAD memory that records the evolving CAD state and allows agents to retrieve relevant historical parameters when needed (3.5).

3.1 PROBLEM FORMULATION

Given a single RGB image $\mathbf{I} \in \mathbb{R}^{H\times W\times 3}$ depicting an object, our goal is to reconstruct the object as a part-aware parametric CAD model $\mathcal{O} = (\mathcal{B}, \mathcal{Q})$, where $\mathcal{B} = \{B_1, \ldots, B_N\}$ is a list of B-Rep shapes representing the object's components, and $\mathcal{Q} = \{q_1, \ldots, q_N\}$ contains the corresponding CadQuery programs. Each program q_i specifies the geometry and spatial placement of the i-th component, and is executed by a CAD kernel to produce its shape:

$$B_i = \text{KernelExecute}(q_i). \tag{1}$$

Together, the generated components form a coherent reconstruction, while their CadQuery programs preserve parametric editability and construction logic for additional downstream CAD operations.

3.2 COMPONENT DECOMPOSING

To avoid the ambiguity and complexity of synthesizing an entire object in a single step, we first employ a VLM to decompose the object depicted in the input image into an ordered sequence of CAD-meaningful components: $\mathcal{P} = [(p_1, b_1), \ldots, (p_N, b_N)]$, where p_i denotes the label of the i-th target component (e.g., `seat_panel`), and b_i is a brief description capturing its semantic and structural characteristics. Each p_i represents either an individual physical part or a coherent subassembly. For occluded or unobserved regions, the VLM combines the available visual evidence with learned object priors to infer plausible missing components and their main forms.

Rather than relying solely on visual appearance, our decomposition reflects practical CAD workflows in both granularity and construction order. Elements with shared geometric structures, such as symmetry or repetition, are treated as a single modeling unit, while the sequence $\mathcal{P}$ arranges all units according to their dependencies. Specifically, structural parts are constructed earlier to establish an

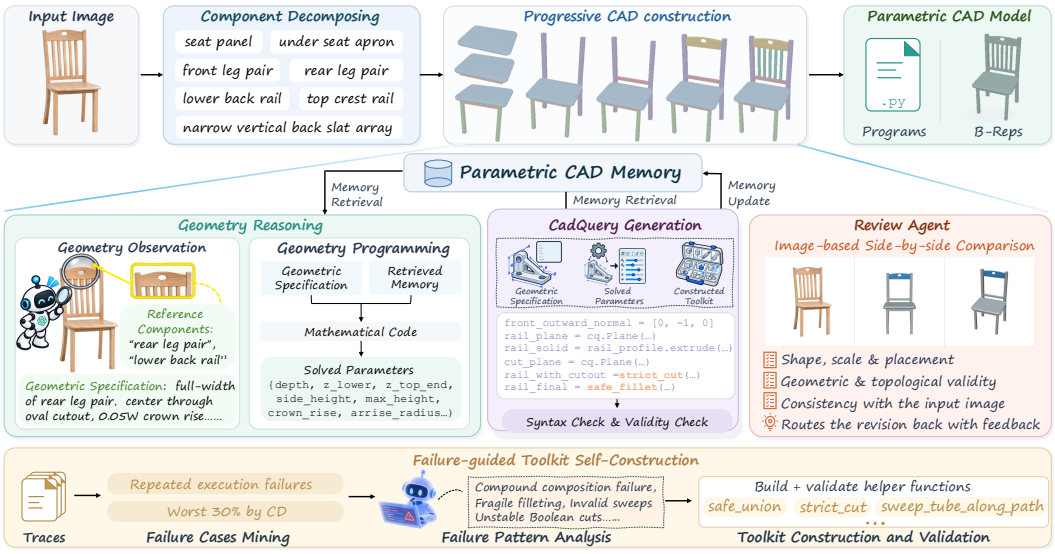


Figure 1: **Overview of CADForge.** Given a single image, CADForge decomposes the object into several components and constructs them progressively. For each component, geometry observation identifies reference components and CAD-relevant constraints, while geometry programming translates them into key modeling parameters. A code generator then synthesizes executable CadQuery code, and a review agent evaluates the result and provides targeted feedback. Throughout the process, a compact parametric CAD memory maintains essential CAD state for on-demand retrieval, while a failure-guided toolkit offers reusable helpers for robust code generation.

overall geometric scaffold, after which dependent parts are shaped with reference to the existing geometry and placed according to attachment, embedding, or alignment relationships (as visualized in Fig. 10). In this way, CADForge derives a compact, structured blueprint that preserves the object's underlying construction logic and provides global guidance for progressive CAD reconstruction.

3.3 Geometry Reasoning

A key challenge in single-view CAD reconstruction is recovering geometric structure and spatial relationships from incomplete visual information. Existing methods often fail to infer these cues reliably, leading to geometric inconsistencies in the reconstructed model. To address this, we introduce a two-stage geometry reasoning module before CAD code generation, consisting of geometry observation and geometry programming to extract and formalize CAD-relevant geometric cues.

3.3.1 Geometry Observation

For each component p_i, CADForge analyzes the input image I and selects a compact reference set $\mathcal{R}_i$ from previously constructed components. The selection is guided primarily by spatial relationships and shape relevance, favoring parts that are spatially related or structurally analogous to p_i. Using $\mathcal{R}_i$ as geometric anchors, the geometry observation module then produces a CAD-oriented specification to guide subsequent shape modeling and component assembly:

$$g_i = \text{Observe}(I, p_i, b_i, \mathcal{R}_i). \tag{2}$$

Concretely, g_i captures both the target component's intrinsic structure and its geometric relationships with $\mathcal{R}_i$. At the intra-component level, g_i describes its profile, principal orientation, symmetry, and repetition patterns. At the inter-component level, it specifies its pose and scale relative to $\mathcal{R}_i$, together with the attachment, alignment and support relationships, which may include constraints such as parallelism and perpendicularity, as well as relative offsets between components. Unlike implicit visual cues, this explicit and structured representation situates p_i relative to the existing geometry and reduces ambiguity, significantly enhancing the consistency of the generated model.

3.3.2 GEOMETRY PROGRAMMING

To bridge the gap between natural-language geometric descriptions and CAD construction, CADForge further introduces Geometry Programming, a module that translates geometric specifications into executable code for computing key modeling parameters. Instead of reasoning about every aspect of a component, this stage focuses on the essential geometric parameters implied by g_i. For each component p_i, CADForge first retrieves the structured parameters $\Theta_{\mathcal{R}_i}$ of its reference components from the parametric CAD memory (3.5), and then uses these parameters along with g_i to synthesize a lightweight Python program s_i, which is executed to obtain the target parameters θ_i:

$$s_i = \mathrm{ProgramLLM}(g_i, \Theta_{\mathcal{R}_i}), \qquad \theta_i = \mathrm{Exec}(s_i). \tag{3}$$

The resulting θ_i is returned as a structured dictionary that records each parameter's name, meaning, and numerical value, providing explicit and inspectable geometric guidance for CAD shape modeling. By converting high-level geometric information into precise, construction-ready parameters, our proposed two-stage Geometry Reasoning greatly enhances geometric accuracy and cross-component consistency while reducing the inference burden for downstream code generation.

3.4 CADQUERY GENERATION

3.4.1 CODE SYNTHESIS

Given the semantic brief b_i, geometric specification g_i, solved parameters θ_i, and the reference modeling parameters $\Theta_{\mathcal{R}_i}$, we employ an LLM-based code generator to synthesize an executable CadQuery program for the current component:

$$c_i = \mathrm{CodeLLM}(p_i, b_i, g_i, \theta_i, \Theta_{\mathcal{R}_i}). \tag{4}$$

Here, c_i denotes the CadQuery program that constructs p_i, which is compiled together with previously generated programs to produce the updated CAD model. Since Geometry Reasoning has already parameterized and resolved the key geometric relationships, the code generator can devote more modeling capacity to generating appropriate CadQuery operations rather than determining spatial configuration, which leads to more stable code synthesis and more faithful CAD geometry. If the generated program fails to execute or produces CAD output that does not pass the validity checks, a code repairer revises the CadQuery code based on the associated error message.

After successful execution, CADForge renders the updated model from two complementary views, with new components highlighted in different colors, and concatenates them with the input image for review. The Review Agent then checks whether the generated component matches the intended shape, scale, and position. If a failure is detected, it attributes the error to either geometry reasoning or code synthesis and routes the revision back to the corresponding stage with targeted feedback.

3.4.2 FAILURE-GUIDED TOOLKIT SELF-CONSTRUCTION

Although CadQuery provides a powerful procedural framework for CAD modeling, synthesizing reliable CadQuery code remains challenging due to its stateful, selector-driven modeling paradigm and strict geometric validity requirements. Even syntactically correct programs may still fail or produce invalid geometry. To further improve robustness and prevent recurring implementation errors, we propose a failure-guided mechanism for automatically constructing a reusable toolkit.

Specifically, we construct a separate set from the Fusion 360 dataset (Willis et al. (2021)) and run our pipeline on the corresponding rendered images, recording complete generation traces that include intermediate observations, geometry programs, synthesized code, execution results, and review feedback. From these traces, we mine two complementary types of failure cases. The first consists of runs with repeated code execution failures, which reveal CadQuery operations that LLMs struggle to implement reliably. The second includes geometrically inaccurate generations, identified by ranking all cases based on the Chamfer Distance and selecting the worst 30% as low-quality cases. An LLM then analyzes these cases together with their generation traces to summarize recurring failure patterns and identify the underlying CAD operations responsible for them. Based on these summaries, we prompt the LLM to encapsulate frequently used yet error-prone modeling operations into robust, reusable helper functions, such as those for solid composition and hole construction. Only helpers that have been verified as effective are incorporated into the toolkit. As new failure traces are collected, the same process can be repeated to expand the toolkit incrementally. During code synthesis,

Table 1: **Quantitative comparison with baselines**. We report CAD Program Validity Rate (CPVR), CD-L1, HD, F-scores, Silhouette Intersection over Union (SIOU), Normal Consistency (NC). For agentic methods, we report Average Token Consumption (AvgT). The best results are in **bold**.

Method	AvgT↓	CD-L1↓	HD↓	F@0.05↑	F@0.01↑	SIOU↑	NC↑	CPVR↑
MiniMax-M3	–	0.02465	0.13204	0.86503	0.42338	0.66092	0.67174	80.40%
Kimi-K3	–	0.01962	0.09968	0.92051	0.44060	0.71130	0.69883	78.90%
GPT-5.5	–	0.01867	0.10329	0.92096	0.47000	0.71382	0.68976	83.49%
GPT-5.6-Sol	–	0.01735	0.09427	0.93256	0.49162	0.73341	0.71575	85.97%
CAD-Coder	–	0.04500	0.20964	0.66808	0.26644	0.55691	0.57553	**94.59%**
Articraft	236548	0.01759	0.10436	0.93115	0.49598	0.73303	0.68912	62.15%
Procedura	331804	0.01724	0.10032	0.93997	0.46771	0.73180	0.69902	86.84%
CADForge (Ours)	**81613**	**0.01580**	**0.08749**	**0.94616**	**0.52104**	**0.75300**	**0.72332**	93.81%

the code generator can directly invoke relevant helpers in the toolkit for fragile operations instead of reimplementing them from scratch, significantly improving reliability and efficiency.

3.5 Parametric CAD Memory

Since CADForge constructs an object component by component, each generation step may depend on the dimensions, spatial placement, and attachment interfaces of previously generated components. Passing the full construction history to the LLM would introduce substantial redundancy and token overhead. To avoid this, CADForge maintains a lightweight parametric CAD memory, implemented as a nested dictionary in which each accepted component is associated with its important modeling parameters. Specifically, when generating the CadQuery program for p_i, CADForge further abstracts the underlying modeling operations into a structured parameter dictionary $\Theta_i = \{(k_j, v_j)\}_{j=1}^{m_i}$, where each key k_j is a concise parameter name (e.g., `full_width`) and v_j stores its corresponding numerical value. Once p_i is accepted, Θ_i is added to the memory as follows:

$$\mathcal{M}_i = \mathcal{M}_{i-1} \cup (p_i \mapsto \Theta_i). \tag{5}$$

For each component, CADForge queries $\mathcal{M}_i$ with the labels of its reference components and retrieves the corresponding parameter entries to guide Geometry Reasoning and Code Synthesis, thereby grounding the generation in the existing geometry while avoiding redundant information.

4 Experiments

4.1 Settings

Evaluation Set. To comprehensively assess our method, we construct an evaluation set of 200 samples. Among them, 170 samples are collected from Objaverse (Deitke et al. (2023)), Fusion 360 (Willis et al. (2021)), and 3D-Future (Fu et al. (2021)), covering single-part objects, multi-part objects, everyday items, and furniture. These samples are used to evaluate geometric metrics. Unless otherwise specified, all subsequent geometry-related evaluations and ablation studies are conducted on this subset. To further demonstrate the out-of-distribution (OOD) generalization capability of our method, we additionally select 30 web images for perceptual evaluation. The entire evaluation set is strictly disjoint from the data used for toolkit construction.

Baselines. Focusing on the single-view CAD reconstruction task, we primarily evaluate mainstream vision-language models (VLMs) and current open-sourced agentic approaches. The evaluated VLMs include MiniMax-M3 (Lai et al. (2026)), Kimi-K3 (Team et al. (2026)), GPT-5.5, and GPT-5.6-Sol, while the agentic baselines consist of CAD-Coder (Doris et al. (2026a)), Articraft (Zhou et al. (2026)), and the concurrent work Procedura (Lin et al. (2026)).

Metrics. To evaluate the geometric reconstruction quality of different methods, we adopt Chamfer Distance (CD-L1) (Mescheder et al. (2019)), Hausdorff Distance (HD) (Huttenlocher et al. (1993)), F-Score (Tatarchenko et al. (2019)), Silhouette Intersection over Union (SIOU) (Wiles & Zisserman (2019)), and Normal Consistency (NC) (Mescheder et al. (2019)) as our primary metrics. For

Table 2: **Ablation study of three key CADForge components**.

Method	CD-L1↓	HD↓	F@0.05↑	F@0.01↑	SIOU↑	NC↑
w/o component decomposing	0.01769	0.10338	0.92886	0.48224	0.73184	0.70218
w/o geometry reasoning	0.01834	0.10554	0.92253	0.47624	0.72935	0.70094
w/o review agent	0.01941	0.10971	0.91455	0.46345	0.71512	0.69510
CADForge	**0.01580**	**0.08749**	**0.94616**	**0.52104**	**0.75300**	**0.72332**

Table 3: **Ablation study of the CADForge toolkit**. Average Repair Count (Avg.Repair Count) represents the average number of repair iterations required to fix syntax errors in the generated code.

Method	Avg. Tokens↓	CPVR↑	Avg. Repair Count↓	CD-L1↓	HD↓	F@0.05↑	NC↑
w/o toolkit	85943	91.08%	1.68750	0.01729	0.09646	0.93345	0.70758
CADForge	**81613**	**93.81%**	**1.29167**	**0.01580**	**0.08749**	**0.94616**	**0.72332**

agentic methods, we additionally report the average total number of tokens consumed to synthesize a CAD model (AvgT), including both input and output tokens. To assess the ability to generate valid CAD models, we further introduce the CAD Program Validity Rate (CPVR), defined as the proportion of all construction attempts that successfully produce a valid CAD model. Besides these objective metrics, we further conduct VLM-based and human preference evaluations, quantified by the preference rates at the top three ranks. More details are provided in C.1.

Implementation Details. All experiments were conducted on the same server and reproduced according to the official documentation or official open-source repositories. For VLMs such as GPT-5.5 and MiniMax-M3, we repeatedly query the model until it generates executable CadQuery code. All agentic methods use GPT-5.5 as the backbone LLM/VLM. For CADForge, each generation step allows at most five code-repair attempts and three failed review attempts. For models generated by different methods, we focus on their geometric structures. Please refer to C.2 for more specifics.

4.2 Quantitative Comparison

Geometric Evaluation. As shown in Tab. 1, CADForge achieves the best performance across all geometric metrics while maintaining a strong CPVR. It reduces CD-L1 to 0.01580, improving over Procedura by 8.35%, and achieves 0.52104 on the more stringent F@0.01 metric, demonstrating better recovery of fine-grained structures. Improvements in HD, SIOU, and NC further confirm its overall geometric fidelity, while its competitive CPVR indicates reliable generation of valid B-reps. Moreover, CADForge consumes only 81.6k tokens on average, 75.4% fewer than Procedura, achieving a favorable balance among geometric quality, CAD validity, and token efficiency.

Preference Evaluation. To complement geometric metrics with perceptual assessment, we further conduct a VLM-based preference evaluation and report the resulting Rank-k ($k = 1, 2, 3$) preference rates in Fig. 2. Specifically, for each input, we render the outputs of the five compared methods from a common viewpoint and arrange them alongside the input image in a single composite image. Gemini-3.1-Pro-Preview then serves as the VLM judge and selects the top three outputs based on an overall assessment of shape similarity, proportional consistency, and geometric fidelity. CADForge achieves the highest Rank-1 preference rate, indicating its strong performance in structural plausibility, geometric accuracy, and perceptual quality. A human user study is provided in E.3.

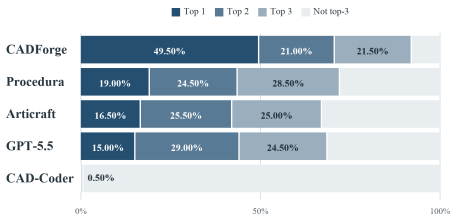


Figure 2: VLM-based preference evaluation across different methods.

4.3 Qualitative Comparison

Fig. 3 presents qualitative comparisons across different methods. CADForge consistently achieves higher reconstruction fidelity and better geometric consistency, particularly for objects with repetitive structures and fine-grained geometric details. For example, on the L-shaped bracket in the first

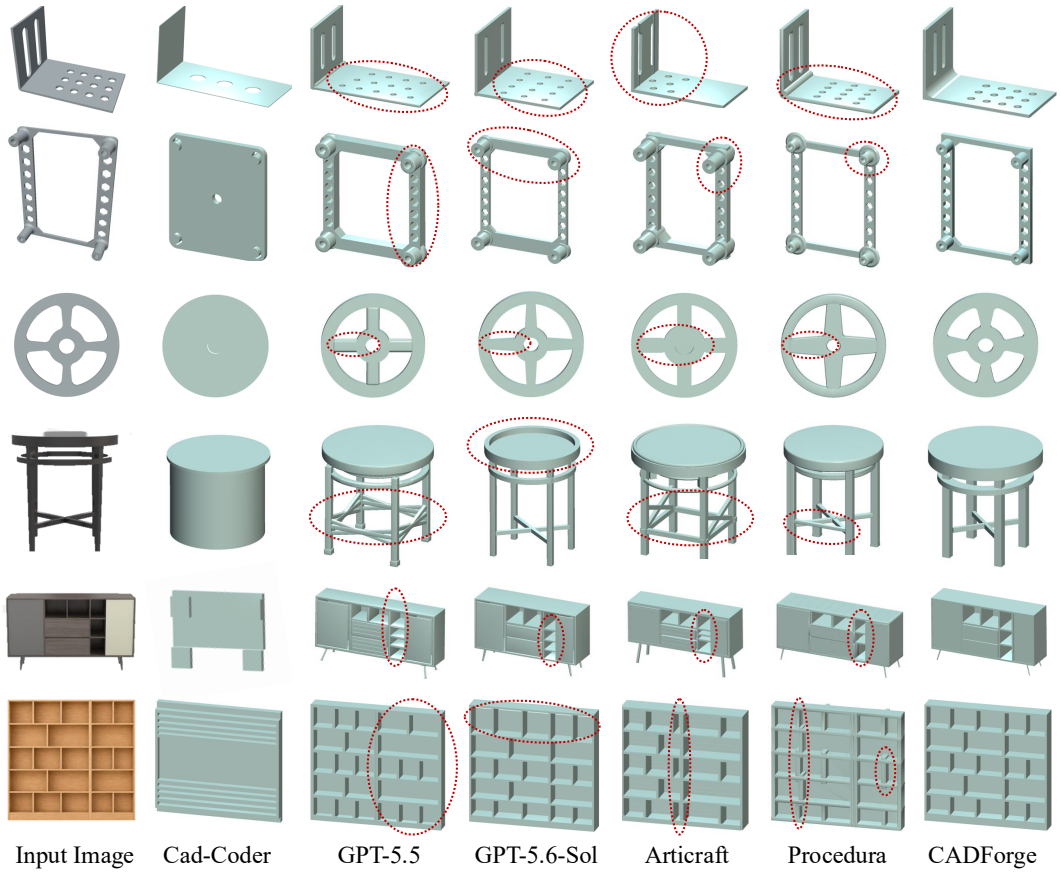


Figure 3: **Qualitative comparison of results generated by different methods**. Red circles highlight regions with geometric inconsistencies relative to the input images.

row, CADForge accurately recovers the number and arrangement of holes, whereas competing methods often miss, distort, or misplace them. Similar structural errors can be observed in the frame and table examples, where other methods introduce distorted joints, incorrect supports, or inconsistent component connections. The differences become more pronounced for complex storage furniture: competing methods frequently produce misaligned partitions, merged components, or incorrect shelf layouts, while CADForge preserves the intended structure. Overall, CADForge produces cleaner, more coherent, and structurally faithful CAD reconstructions across a diverse range of objects.

4.4 Ablation Studies

Ablation on Framework Components. We conduct ablation studies on the three key modules of CADForge: component decomposing, geometry reasoning, and the review agent, with quantitative results reported in Tab. 2 and qualitative comparisons presented in Fig. 4. Experimental results show that removing any module degrades performance, validating the coherence of our unified design and the effectiveness of each stage. As shown in Fig. 4, when component decomposition is replaced with step-wise component selection based on the current partial result, the reconstruction is prone to missing components due to the lack of global planning. Similarly, removing geometry reasoning noticeably compromises geometric accuracy, causing distortions such as incorrect aspect ratios. Without the review agent, the model is less capable of identifying and correcting existing mistakes, often leading to error accumulation over successive generation steps. We further provide an ablation study on geometry programming in the supplementary material.

Ablation on Toolkit. To examine the impact of the failure-guided self-constructed toolkit, we remove it from CADForge for ablation in Tab. 3. Without it, both the CPVR and Average Repair Count deteriorate, suggesting that the code generator more frequently produces non-executable code and requires additional repair iterations, consequently increasing the Average Token consumption.

Table 5: **Performance of CADForge with different backbone models**.

Backbone	CD-L1↓	HD↓	F@0.05↑	F@0.01↑	SIOU ↑	NC ↑
GPT-5.6-Sol	0.01539	0.08740	0.94076	0.56187	0.76068	0.73160
Gemini-3.1-Pro-Preview	0.01620	0.09299	0.93768	0.52581	0.74610	0.71549
GPT-5.5 (Ours)	0.01580	0.08749	0.94616	0.52104	0.75300	0.72332

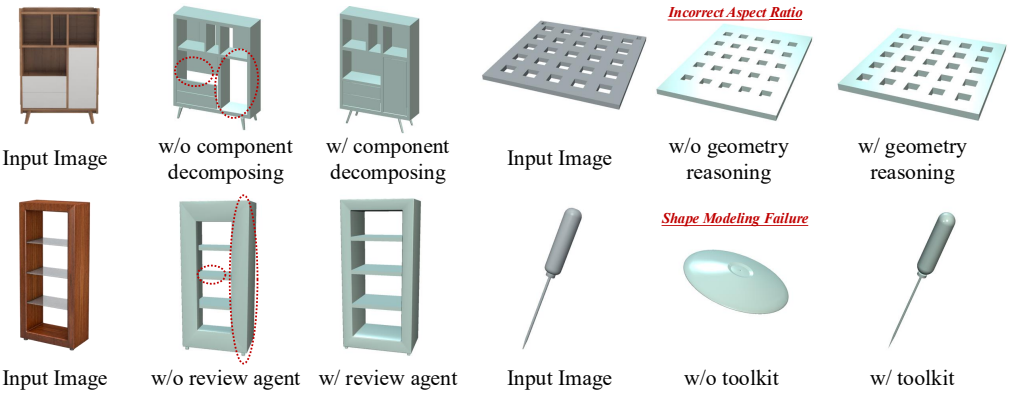


Figure 4: **Qualitative comparison of the ablation studies**, illustrating the effectiveness of component decomposition, geometry reasoning, the review agent, and the self-constructed toolkit.

Moreover, the absence of modeling helpers can limit the code generator's ability to construct some shapes or structures, degrading metrics such as HD. As evidenced in the bottom-right example of Fig. 4, despite multiple rounds of code repair and review, the code generator without toolkit still fails to accurately model the cylindrical component, resulting in a clear mismatch with the input image.

Ablation on Parametric CAD Memory. We further conduct an ablation study of the parametric CAD memory on the 3D-FUTURE subset (Tab. 4). Since CADForge generates models component by component, each new component has to depend on information from previous ones. Thus, completely removing memory without any historical context is not a meaningful ablation. For the variant without our proposed memory, we follow prior methods and provide the full generation histories of the reference components. Besides the geometric metrics, we additionally report the number of tokens consumed per component (Tok./Comp.). While both variants achieve comparable generation performance, our memory substantially reduces token consumption, demonstrating that it effectively compresses long generation histories while preserving the needed geometric information.

Table 4: **Ablation on the CAD memory**.

	Tok./Comp.↓	CD-L1↓	HD↓
w/o Memory	45696	0.01595	0.10137
w/ Memory	27225	0.01601	0.09332

Ablation on VLM Backbones. To assess the generalizability of our framework, we replace the backbone of CADForge with GPT-5.6-Sol and Gemini-3.1-Pro-Preview and benchmark it under different model configurations. The results in Tab. 5 indicate that CADForge maintains strong results across different backbone models, demonstrating that its effectiveness is not tied to a specific foundation model. Importantly, CADForge with GPT-5.6-Sol delivers the best overall performance, implying that advances in foundation model capability can be effectively translated into further gains within our agentic framework. We additionally provide visual comparisons in Fig. 11.

5 Conclusion

We present CADForge, a training-free agentic framework for single-view CAD reconstruction. CADForge progressively reconstructs CAD models by decomposing objects into components, explicitly reasoning about their geometry, and iteratively refining the generated programs, while improving robustness and efficiency through a failure-guided toolkit and compact parametric memory. Experiments show that CADForge achieves higher reconstruction fidelity with lower token consumption than existing baselines. Overall, our results demonstrate the effectiveness of CADForge for accurate and efficient parametric CAD reconstruction from a single-view image.

REFERENCES

Md Ferdous Alam and Faez Ahmed. Gencad: Image-conditioned computer-aided design generation with transformer-based contrastive representation and diffusion priors. *arXiv preprint arXiv:2409.16294*, 2024.

Jesse Barkley, Rumi Loghmani, and Amir Barati Farimani. Cadsmith: Multi-agent cad generation with programmatic geometric validation. *arXiv preprint arXiv:2603.26512*, 2026.

Federico Campi, Claudio Favi, Michele Germani, and Marco Mandolini. Cad-integrated design for manufacturing and assembly in mechanical design. *International Journal of computer integrated manufacturing*, 35(3):282–325, 2022.

Cheng Chen, Jiacheng Wei, Tianrun Chen, Chi Zhang, Xiaofeng Yang, Shangzhan Zhang, Bingchen Yang, Chuan-Sheng Foo, Guosheng Lin, Qixing Huang, and Fayao Liu. Cadcrafter: Generating computer-aided design models from unconstrained images. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11073–11082, 2025a. doi: 10.1109/CVPR52734.2025.01034.

Tianrun Chen, Chunan Yu, Yuanqi Hu, Jing Li, Tao Xu, Runlong Cao, Lanyun Zhu, Ying Zang, Yong Zhang, Zejian Li, and Lingyun Sun. Img2cad: Conditioned 3-d cad model generation from single image with structured visual geometry. *IEEE Transactions on Industrial Informatics*, 21 (11):8539–8549, 2025b. doi: 10.1109/TII.2025.3584476.

Bingquan Dai, Luo Li, Qihong Tang, Jie Wang, Xinyu Lian, Hao Xu, Minghan Qin, Xudong XU, Bo Dai, Haoqian Wang, Zhaoyang Lyu, and Jiangmiao Pang. Meshcoder: Llm-powered structured mesh code generation from point clouds. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 8917–8954. Curran Associates, Inc., 2025. doi: 10.52202/085713-0301. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/0d094d83efa5ea99f86527ac7207e023-Paper-Conference.pdf.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13142–13153, June 2023.

Anna C Doris, Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. Cad-coder: An open-source vision-language model for computer-aided design code generation. *Journal of Mechanical Design*, 148(7):071702, 2026a.

Anna C Doris, Jacob Thomas Sony, Ghadi Nehme, Era Syla, Amin Heyrani Nobari, and Faez Ahmed. Cadbench: A multimodal benchmark for ai-assisted cad program generation. *arXiv preprint arXiv:2605.10873*, 2026b.

Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *International Journal of Computer Vision*, 129 (12):3313–3337, 2021.

Yandong Guan, Xilin Wang, XiMing Xing, Jing Zhang, Dong Xu, and Qian Yu. Cad-coder: Text-to-cad generation with chain-of-thought and geometric reward. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 59765–59789. Curran Associates, Inc., 2025. doi: 10.52202/085713-1999. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/564a224e88f2490f3c1deaae877d37e3-Paper-Conference.pdf.

Ziniu Hu, Ahmet Iscen, Aashi Jain, Thomas Kipf, Yisong Yue, David A Ross, Cordelia Schmid, and Alireza Fathi. Scenecraft: An llm agent for synthesizing 3d scenes as blender code. In *ICML*, pp. 19252–19282, 2024.

Daniel P Huttenlocher, Gregory A Klanderman, William Rucklidge, et al. Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9): 850–863, 1993.

Soslan Kabisov, Vsevolod Kirichuk, Andrey Volkov, Marina Barannikov, Gennadiy Savrasov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. Cadreasoner: Iterative program editing for cad reverse engineering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Findings*, pp. 6143–6153, June 2026.

Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4713–4722, 2024a. doi: 10.1109/CVPR52733.2024.00451.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 7552–7579. Curran Associates, Inc., 2024b. doi: 10.52202/079017-0242. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0e5b96f97c1813bb75f6c28532c2ecc7-Paper-Conference.pdf.

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

Xunhao Lai, Weiqi Xu, Yufeng Yang, Qiaorui Chen, Yang Xu, Lunbin Zeng, Xiaolong Li, Haohai Sun, Haichao Zhu, Vito Zhang, et al. Minimax sparse attention. *arXiv preprint arXiv:2606.13392*, 2026.

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. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573, 2025a. doi: 10.1109/CVPR52734.2025.01730.

Xingang Li, Yuewan Sun, and Zhenghui Sha. Llm4cad: Multimodal large language models for three-dimensional computer-aided design generation. *Journal of Computing and Information Science in Engineering*, 25(2):021005, 2025b.

Xueyang Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Cad translator: An effective drive for text to 3d parametric computer-aided design generative modeling. In *Proceedings of the 32nd ACM International Conference on Multimedia*, MM '24, pp. 8461–8470, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706868. doi: 10.1145/3664647.3681549. URL https://doi.org/10.1145/3664647.3681549.

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. In *International Conference on Learning Representations*, volume 2026, pp. 94278–94298, 2026a.

Yuan Li, Cheng Lin, Yuan Liu, Xiaoxiao Long, Chenxu Zhang, Ningna Wang, Xin Li, Wenping Wang, and Xiaohu Guo. Caddreamer: Cad object generation from single-view images. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21448–21457, 2025c. doi: 10.1109/CVPR52734.2025.01998.

Yuanbo Li, Dule Shu, Yanying Chen, Matt Klenk, and Daniel Ritchie. Pllm: Pseudo-labeling large language models for cad program synthesis. *arXiv preprint arXiv:2602.12561*, 2026b.

Youtian Lin, Yikang Yang, Zhanpeng Hu, Mengqi Zhou, Feihu Zhang, Xun Cao, Jiaheng Liu, and Yao Yao. Procedura: Agentic 3d modeling with procedural control. *arXiv preprint arXiv:2608.26238*, 2026.

Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2cad: Reverse engineering cad models from 3d point clouds. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3763–3772, 2024. doi: 10.1109/CVPR52733.2024.00361.

Sining Lu, Guan Chen, Nam Anh Dinh, Itai Lang, Ari Holtzman, and Rana Hanocka. Ll3m: Large language 3d modelers. *arXiv preprint arXiv:2508.08228*, 2025.

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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27144–27153, 2024. doi: 10.1109/CVPR52733.2024.02564.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4455–4465, 2019. doi: 10.1109/CVPR.2019.00459.

K Lalit Narayan, K Mallikarjuna Rao, and MMM Sarcar. *Computer aided design and manufacturing*. PHI Learning Private Limited, 2013.

James Noeckel, Benjamin Jones, Adriana Schulz, and Brian Curless. View2cad: Reconstructing view-centric cad models from single rgb-d scans. *arXiv preprint arXiv:2504.04000*, 2025.

Vladislav Pyatov, Gleb Bobrovskikh, Saveliy Galochkin, Nikita Boldyrev, Oleg Voynov, Alexander Filippov, Gonzalo Ferrer, Peter Wonka, and Evgeny Burnaev. Cadfs: A big cad program dataset and framework for computer-aided design with large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10176–10186, June 2026.

Dacheng Qi, Chenyu Wang, Jingwei Xu, Tianzhe Chu, Zibo Zhao, Wen Liu, Wenrui Ding, Yi Ma, and Shenghua Gao. Pointer-cad: Unifying b-rep and command sequences via pointer-based edges & faces selection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17377–17387, June 2026.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-recode: Reverse engineering cad code from point clouds. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9801–9811, 2025. doi: 10.1109/ICCV51701.2025.00914.

Emanuel Sachs, Michael Cima, and James Cornie. Three-dimensional printing: rapid tooling and prototypes directly from a cad model. *CIRP annals*, 39(1):201–204, 1990.

Liviu-Iacob Scurtu, BODI Ștefan, and Mihai Dragomir. Optimization methods applied in cad based furniture design. *ACTA TECHNICA NAPOCENSIS-Series: APPLIED MATHEMATICS, MECHANICS, and ENGINEERING*, 58(4), 2015.

Chunyi Sun, Junlin Han, Weijian Deng, Xinlong Wang, Zishan Qin, and Stephen Gould. 3d-gpt: Procedural 3d modeling with large language models. In *2025 International Conference on 3D Vision (3DV)*, pp. 1253–1263, 2025. doi: 10.1109/3DV66043.2025.00119.

Shiyu Tan, Zixuan Zhao, Hao Gao, Zhiheng Chen, Xiaolong Yin, and Enya Shen. Img2cadseq: Image-to-cad generation via sequence-based diffusion. *arXiv preprint arXiv:2605.13293*, 2026.

Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 3400–3409. IEEE, 2019.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Puyi Wang, Yuhao Wang, Linjie Li, Zhengyuan Yang, Kevin Qinghong Lin, Yangguang Li, and Yu Cheng. Scenecode: Executable world programs for editable indoor scenes with articulated objects. *arXiv preprint arXiv:2605.19587*, 2026.

Olivia Wiles and Andrew Zisserman. Learning to predict 3d surfaces of sculptures from single and multiple views. *International Journal of Computer Vision*, 127(11):1780–1800, 2019.

Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: a dataset and environment for programmatic cad construction from human design sequences. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459818. URL https://doi.org/10.1145/3450626.3459818.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6752–6762, 2021. doi: 10.1109/ICCV48922.2021.00670.

Hongchi Xia, Xuan Li, Zhaoshuo Li, Qianli Ma, Jiashu Xu, Ming-Yu Liu, Yin Cui, Tsung-Yi Lin, Wei-Chiu Ma, Shenlong Wang, et al. Sage: Scalable agentic 3d scene generation for embodied ai. *arXiv preprint arXiv:2602.10116*, 2026.

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

Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. *arXiv preprint arXiv:2207.04632*, 2022.

Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable cad model generation. *arXiv preprint arXiv:2307.00149*, 2023.

Zhifei Yang, Keyang Lu, Chao Zhang, Jiaxing Qi, Hanqi Jiang, Ruifei Ma, Shenglin Yin, Yifan Xu, Mingzhe Xing, Zhen Xiao, et al. Mmgdreamer: Mixed-modality graph for geometry-controllable 3d indoor scene generation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 9391–9399, 2025.

Zhifei Yang, Guangyao Zhai, Keyang Lu, YuYang Yin, Chao Zhang, Zhen Xiao, Jieyi Long, Nassir Navab, and Yikai Wang. Flowscene: Style-consistent indoor scene generation with multimodal graph rectified flow. *arXiv preprint arXiv:2603.19598*, 2026.

Zhanwei Zhang, kaiyuan liu, Junjie Liu, Wenxiao Wang, Binbin Lin, Liang Xie, Chen Shen, and Deng Cai. Geocad: Local geometry-controllable cad generation with large language models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 130989–131014. Curran Associates, Inc., 2025. doi: 10.52202/085713-4363. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/bdd4cb9ca5cf3bed4fa66e0677a0ab6a-Paper-Conference.pdf.

Matt Zhou, Ruining Li, Xiaoyang Lyu, Zhaomou Song, Zhening Huang, Chuanxia Zheng, Christian Rupprecht, Andrea Vedaldi, and Shangzhe Wu. Articraft: An agentic system for scalable articulated 3d asset generation. *arXiv preprint arXiv:2605.15187*, 2026.

Mengqi Zhou, Yuxi Wang, Jun Hou, Shougao Zhang, Yiwei Li, Chuanchen Luo, Junran Peng, and Zhaoxiang Zhang. Scenex: Procedural controllable large-scale scene generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 10806–10814, 2025.

A AI USE STATEMENT

Large language models and AI tools were used for two purposes. First, they assisted with language polishing, including wording, grammar, and sentence structure. All AI-assisted revisions were carefully reviewed and proofread by the authors to ensure factual correctness, preserve the intended meaning, and avoid hallucinated or unsupported statements. Second, as CADForge is an agentic framework, VLMs were used as components of the proposed pipeline and for perceptual assessment, as described in the paper. All such uses followed human-defined procedures and were subject to human supervision and verification. The research design, experimental protocol, result interpretation, and scientific conclusions were determined by the authors. The authors take full responsibility for the final content of this work, including text, claims and conclusions.

B EXPERIMENTAL DETAILS

B.1 EVALUATION SET

To assess single-view CAD reconstruction across diverse object categories, we curate an evaluation set of 200 samples, including 170 objects from Objaverse (Deitke et al. (2023)), Fusion 360 (Willis et al. (2021)), and 3D-Future (Fu et al. (2021)), as well as 30 web images. The 170 samples span simple and complex CAD parts, everyday objects, and furniture, and are used for geometric evaluation. For Objaverse and Fusion 360, we follow the data-processing protocol of CADBench (Doris et al. (2026b)). Each object is rendered from one of several representative viewpoints rather than a fixed canonical view. The 30 web images are collected to assess out-of-distribution generalization through qualitative comparison and perceptual assessment. Examples are shown in Fig. 5.

C BASELINES

We primarily compare CADForge with two categories of methods that are directly executable under our evaluation setting: (1) general-purpose VLMs capable of generating CadQuery programs from images, including MiniMax-M3, Kimi-K3, GPT-5.5, and GPT-5.6-Sol; and (2) recent CAD-related agentic or programmatic 3D modeling methods, including CAD-Coder, Articraft, and Procedura. We reproduce all open-sourced methods following their official implementations and use the same backbone model whenever applicable.

Several recent single-view CAD reconstruction methods are closely related to our task. However, some of them such as Img2CADSeq (Tan et al. (2026)), CADCrafter (Chen et al. (2025a)), and Img2CAD (Chen et al. (2025b)) do not provide publicly available code or inference checkpoints, while other works, such as CADDreamer (Li et al. (2025c)), adopt a different setting from ours. Therefore, we do not include them in the main quantitative comparison.

C.1 METRICS

C.1.1 METRIC COMPUTATION

We first put each generated output and its corresponding ground truth in the same canonical object-centric coordinate frame, where the x-axis points to the object's right, the y-axis points from the object's canonical front toward its back, and the z-axis points upward. We then translate both to the origin based on their geometric centers and apply scale normalization. Then we uniformly sample $N_s = 50{,}000$ points from the surfaces of both the predicted model and the ground-truth model. Let $P = \{(\mathbf{p}_i, \mathbf{n}_i^P)\}_{i=1}^{N_s}, G = \{(\mathbf{g}_j, \mathbf{n}_j^G)\}_{j=1}^{N_s}$ denote the sampled surface points and their corresponding face normals. For a point $\mathbf{x}$ and a point set Y, we define its nearest-neighbor distance as

$$d(\mathbf{x}, Y) = \min_{\mathbf{y} \in Y} \|\mathbf{x} - \mathbf{y}\|_2, \tag{6}$$

and its nearest-neighbor index as

$$\pi_Y(\mathbf{x}) = \arg\min_{\mathbf{y} \in Y} \|\mathbf{x} - \mathbf{y}\|_2. \tag{7}$$



Figure 5: **Representative input images from our evaluation set**.

Chamfer Distance (CD-L1). We adopt the L1 variant of Chamfer Distance and measure the average bidirectional discrepancy between the predicted and ground-truth surfaces:

$$\mathrm{CD}(P,G) = \frac{1}{2}\left[\frac{1}{|P|}\sum_{\mathbf{p}\in P} d(\mathbf{p},G) + \frac{1}{|G|}\sum_{\mathbf{g}\in G} d(\mathbf{g},P)\right],$$

where $d(\mathbf{x},Y) = \min_{\mathbf{y}\in Y}\|\mathbf{x}-\mathbf{y}\|_2$. The first term measures the distance from the generated surface to the ground truth, while the second measures how well the generated model covers the ground-truth surface. Lower values indicate greater geometric similarity.

Hausdorff Distance (HD). While CD measures the average surface discrepancy, Hausdorff Distance captures the largest bidirectional deviation:

$$\mathrm{HD}(P,G) = \max\left\{\max_{\mathbf{p}\in P} d(\mathbf{p},G),\ \max_{\mathbf{g}\in G} d(\mathbf{g},P)\right\}. \tag{8}$$

HD is particularly sensitive to severe local errors, such as an excessively long component, a detached part, or a missing region. Since it is computed from finite surface samples, the reported value is a sampled approximation of the exact mesh-to-mesh Hausdorff Distance. Lower values are better.

F-score. Given a normalized distance threshold τ, we compute geometric precision and recall as

$$\mathrm{Prec}_\tau(P,G) = \frac{1}{|P|}\sum_{\mathbf{p}\in P}\mathbb{I}[d(\mathbf{p},G)<\tau], \quad \mathrm{Rec}_\tau(P,G) = \frac{1}{|G|}\sum_{\mathbf{g}\in G}\mathbb{I}[d(\mathbf{g},P)<\tau], \tag{9}$$

where $\mathbb{I}[\cdot]$ is the indicator function. Their harmonic mean gives

$$\mathrm{F}_\tau = \frac{2\,\mathrm{Prec}_\tau\,\mathrm{Rec}_\tau}{\mathrm{Prec}_\tau + \mathrm{Rec}_\tau}. \tag{10}$$

When both precision and recall are zero, the F-score is set to zero. We report $F@0.01$ and $F@0.05$, with thresholds set to 1% and 5% of the normalized bounding-box diagonal, respectively. $F@0.01$ emphasizes fine-scale geometric accuracy, whereas $F@0.05$ reflects structural agreement under a more tolerant threshold.

Silhouette Intersection over Union (S-IoU). To evaluate projected shape consistency, we sample $N_v = 100{,}000$ surface points from each aligned mesh and project them onto a shared image plane from four predefined orthographic views:

$$\mathcal{V} = \{\text{front}, \text{side}, \text{top}, \text{oblique}\}. \tag{11}$$

The front, side, and top views correspond to projections onto the xz, yz, and xy planes, respectively. The oblique view is obtained by applying a 45° yaw rotation before projection. The projected points are rasterized into binary occupancy masks at a resolution of 384×384. We apply the same rasterization and mask-processing procedure to both the prediction and the ground truth.

Let S_v^P and S_v^G denote the predicted and ground-truth silhouette masks under view v. The silhouette IoU for that view is

$$\mathrm{IoU}_v = \frac{|S_v^P \cap S_v^G|}{|S_v^P \cup S_v^G|}. \tag{12}$$

The final S-IoU is the unweighted mean over all evaluation views:

$$\text{S-IoU}(P, G) = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \text{IoU}_v \,. \tag{13}$$

Unlike point-wise surface distances, S-IoU directly evaluates consistency in the projected outer shape and visible spatial occupancy. It is therefore sensitive to differences in global proportions, component coverage, and projected layout. Higher values indicate better silhouette fidelity.

Normal Consistency. To measure local surface orientation, we compare the normal at each sampled point with the normal of its nearest neighbor on the other surface. Bidirectional normal consistency is computed as

$$\text{NC}(P, G) = \frac{1}{2} \left[\frac{1}{|P|} \sum_{\mathbf{p}_i \in P} \left| \mathbf{n}_i^P \cdot \mathbf{n}^G_{\pi_G(\mathbf{p}_i)} \right| + \frac{1}{|G|} \sum_{\mathbf{g}_j \in G} \left| \mathbf{n}_j^G \cdot \mathbf{n}^P_{\pi_P(\mathbf{g}_j)} \right| \right] . \tag{14}$$

The absolute dot product makes the metric robust to inconsistent face winding introduced during CAD-to-mesh conversion, treating opposite normal signs as equivalent. Normal Consistency ranges from 0 to 1, with higher values indicating more consistent local surface orientation.

CAD Program Validity Rate. For CAD generation, reliably producing valid B-reps is particularly important. Final generation success alone, however, does not fully characterize execution stability for agentic methods, since failed programs can often be iteratively repaired or regenerated until a valid result is eventually obtained. Two methods may therefore achieve the same final success rate while requiring substantially different numbers of failed executions and repair attempts. To capture this behavior, we additionally record all code execution attempts during inference and report the CAD Program Validity Rate (CPVR) as a complementary measure of execution stability. Because different methods adopt inherently different generation and repair workflows, defining a strictly workflow-independent notion of an "attempt" is difficult. We therefore follow each method's native execution procedure rather than artificially imposing a unified repair protocol. Specifically, for VLMs and CAD-Coder, each generated program is counted as one execution attempt; for CADForge and Procedura, each component-level construction attempt, including the initial generation and subsequent repair attempts, is counted separately; and for Articraft, each compilation attempt along its generation trajectory is counted as one attempt. We define

$$\text{CPVR} = \frac{N_{\text{success}}}{N_{\text{exec}}} = 1 - \frac{N_{\text{fail}}}{N_{\text{exec}}}, \tag{15}$$

where an attempt is considered successful if it executes correctly and produces a valid geometry. If a model can consistently produce valid CAD programs at each construction step, this suggests stronger intrinsic CAD modeling capability. Thus, a higher CPVR indicates more stable code generation and less dependence on iterative code repair.

C.1.2 Preference Evaluation Criteria

The aforementioned metrics, such as CD-L1, HD, and SIOU, primarily evaluate generated CAD models from a geometric perspective by measuring geometric distances, shape discrepancies, and spatial overlap. However, they do not fully capture perceptual quality or human preferences regarding the overall appearance and structural plausibility of the generated models. To complement these geometric metrics, we conduct both VLM-based and human preference evaluations. For these evaluations, we compare CADForge with CAD-Coder, two other agentic methods, Procedura and Articraft, and GPT-5.5, which serves as the backbone model of our framework. Both evaluations follow a shared set of criteria covering overall fidelity, global geometry, component-level structure, local CAD features, and potential structural defects, as detailed below.

Prompt for Preference Evaluation

System Prompt. You are a rigorous and impartial evaluator of image-to-CAD reconstruction, whose goal is to reconstruct a corresponding CAD model from a reference image. Judge the candidates internally and return only the requested top-three ranking line.

Evaluation Instruction. Rank the three best CAD candidates by their overall fidelity to the reference image.
Considering the reference image, assess how effectively each candidate reconstructs the depicted object based on the following factors:

- overall similarity to the reference image and the aesthetic quality of the reconstructed model;
- overall silhouette, dominant shape, and global proportions;
- the number, shape, relative size, position, orientation, arrangement, and connection of major components;
- important geometric relationships, including alignment, spacing, symmetry, attachment, support, and containment;
- visible local CAD features such as holes, slots, openings, recesses, and protrusions;
- missing major structures, unsupported additions, floating parts, and severe unintended intersections.

Image Layout.

- The top-left image is the reference image. It is used only as the target and must not be ranked.
- The five CAD candidates are labeled by position in reading order: `a` is top-center, `b` is top-right, `c` is bottom-left, `d` is bottom-center, and `e` is bottom-right.

Output Rules.

- Select and rank only the three best candidates, with no ties.
- Do not provide analysis, explanations, headings, markdown, or any other text.
- Return exactly one line in the following format:

```
TOP 3:  1=a, 2=b, 3=c
```

Replace the letters with the actual ordering. Use three distinct letters from `a--e`.

For the VLM-based evaluation, we arrange the reference image and the rendered outputs of the five evaluated methods into a 2×3 image sheet and present it to Gemini-3.1-Pro-Preview. Following the evaluation criteria, the model assesses the candidates holistically and selects the exact Rank-1, Rank-2, and Rank-3 results that best reconstruct the object depicted in the reference image.

For the human user study, we develop a web-based interface that presents the same reference image and five rendered results in each trial, as shown in Fig. 6. Method identities are anonymized, and the display order is randomized to reduce potential bias. Participants assess the candidates according to the guidelines and select their exact Rank-1, Rank-2, and Rank-3 choices for each case.

C.2 Implementation Details

All experiments are conducted on a server equipped with two Intel Xeon Platinum 8350C CPUs and an NVIDIA A800 GPU with 80 GB of memory. For general-purpose VLMs, we follow the official documentation and recommended inference settings. Each model is provided with the input image and instructed to generate the corresponding CadQuery code. Since direct VLM generation and CAD-Coder may occasionally produce non-executable programs, we repeatedly query these methods until an executable CAD program is obtained, ensuring that every evaluation sample has a valid output for geometric comparison.

For all baselines, we use the officially released checkpoints and follow the default configurations provided in their respective repositories. For agentic methods, including Procedura, ArtiCraft, and CADForge, we use GPT-5.5 as the backbone model and retain their native iterative regeneration and repair mechanisms. For CADForge, we allow up to five code-repair attempts and three review-triggered revisions for each component-generation step.

For implementation convenience, we also equip the Geometry Program Agent with a small deterministic library of reusable utility functions for common geometric computations in CAD. These

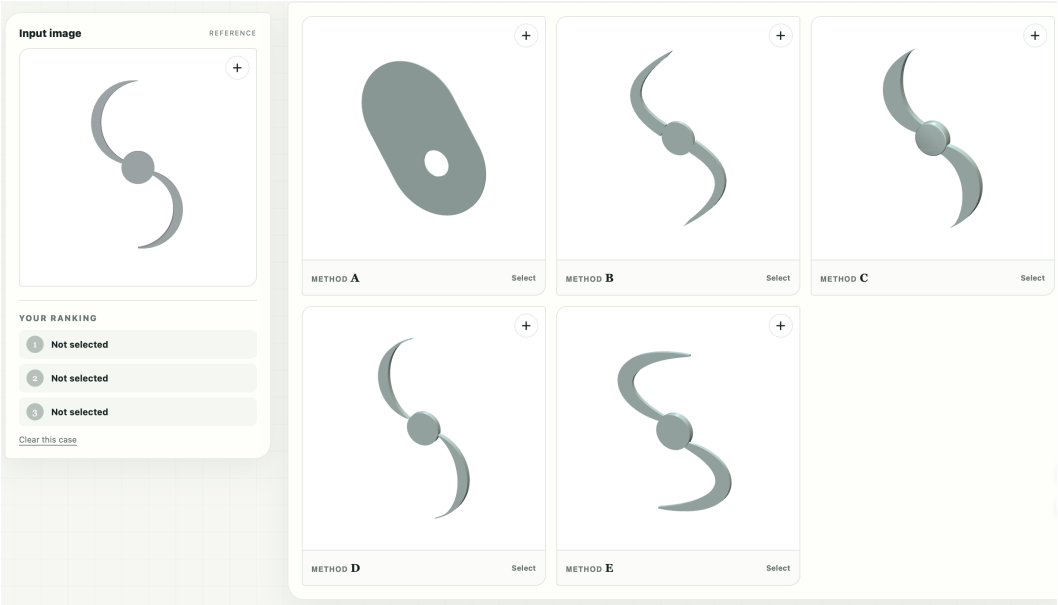


Figure 6: **Interface for the user study.** In each trial, participants compare the input image with five anonymized and randomly ordered results, and select their Rank-1, Rank-2, and Rank-3 choices.

utilities are simple functional wrappers, introduce no additional learned components, and can be equivalently implemented using local mathematical operations.

After each successful component-level CadQuery execution, CADForge exports the updated model in STEP, STL, and GLB formats and use the GLB representation for visual review. The model is rendered off-screen with VTK from two viewpoints: a front view at an elevation of 10° and an azimuth of -90°, and a front-biased oblique view at an elevation of 30° and an azimuth of -70°. The camera is directed toward the center of the model's bounding box with a fixed z-up convention, while its distance is adjusted according to the model extent. We use opaque Phong-shaded surfaces, a white background, explicit feature edges, and three-point illumination without cast shadows. Previously generated components are rendered in gray, whereas the component introduced at the current step is highlighted in blue. Finally, the reference image and the two CAD renderings are resized without cropping or distortion and concatenated into a single review image for direct visual comparison.

For toolkit construction, we use a separate set of 60 cases from the Fusion 360 dataset. Each generated helper is validated for syntactic correctness and retained only if its application improves reconstruction performance on the corresponding failure cases. Through this process, CADForge develop a final toolkit consisting of 10 tools.

For qualitative visualization, we adopt a unified CAD-style rendering protocol across all methods. For each input image, the generated models are rendered using the same camera configuration, white background, lighting setup, and soft-aqua material color with an RGB value of ((189, 214, 210)). For methods such as Articraft that may generate textured or colored meshes, we remove the original textures and materials before rendering. This allows the comparison to focus on geometric structure, including the overall shape, component layout, proportions, holes, cutouts, and local CAD features, rather than differences in appearance or rendering style.

C.3 Introduction of the Toolkit

We implement the self-constructed toolkit as a lightweight library of reusable Python helper functions that encapsulate common yet failure-prone CadQuery operations, such as primitive placement, path-based tubular construction and solid composition. The helpers are prepended to the execution namespace of each generated program, allowing the Code Agent to invoke them directly without additional imports. To expose the toolkit to the LLM, we include a compact API manifest in the prompts for initial code generation, component synthesis, and code repair. The manifest provides

each helper's name, function signature, intended functionality, and concise usage examples, together with lightweight guidance for function selection. Since only this compact interface description is exposed to the Code Agent, incorporating new tools introduces only negligible additional token overhead. The Code Agent is encouraged to invoke a helper whenever it matches the required modeling operation, while retaining native CadQuery primitives for geometries not covered by the toolkit. In this way, the toolkit abstracts away fragile low-level implementation details while preserving the flexibility of general CadQuery modeling.

Below, we present a simple helper and its specific implementation. Rounded boxes are a common pattern in CAD modeling and are typically constructed by creating a box and filleting its edges. Directly asking the Code Agent to repeatedly synthesize low-level operation chains such as `.box(...).edges(...).fillet(...)` can be fragile, as errors may arise from incorrect API composition, inappropriate fillet radii, or kernel-level fillet failures. CADForge therefore introduces a reusable `rounded_box` helper that encapsulates this operation sequence, constrains the fillet radius to a geometry-aware safe range, and falls back to the original valid solid when filleting fails. This helper provides a robust primitive for common mildly rounded components such as panels, drawer fronts, cabinet doors, and seat cushions.

Listing 1: Implementation of the `rounded_box` and `safe_fillet` helpers.

```
def safe_fillet(obj, radius, selector=None):
    try:
        radius = float(radius)
        if radius <= 0:
            return obj
        target = obj.edges(selector) if selector else obj.edges()
        return target.fillet(radius)
    except Exception:
        return obj


def rounded_box(center, size, radius):
    solid = box_at(center, size)
    sx, sy, sz = [abs(float(v)) for v in size]
    limit = max(1e-6, min(sx, sy, sz) * 0.45)
    return safe_fillet(
        solid, min(abs(float(radius)), limit)
    )
```

D Extension and Application

D.1 Texture Generation and Articulation Modeling

Since CADForge represents an object as part-wise B-rep geometry with executable modeling code, its output can be readily augmented with additional appearance and articulation attributes. By incorporating two LLM-based agents for texture assignment and articulation modeling, this representation can be extended to support textured and articulated 3D asset generation. Specifically, the texture agent assigns PBR materials to individual parts, while the articulation agent identifies movable parts and predicts their joint types, motion axes, and motion ranges. As shown in Fig. 8, CADForge more faithfully preserves fine-grained part structures than ArtiCraft, such as the number and arrangement of shelves, while maintaining proper connectivity between movable parts and their supporting structures. These results further demonstrate the flexibility of CADForge's structured representation in supporting appearance and articulation modeling beyond geometric reconstruction.

D.2 Editing Flexibility

The code-based representation of CADForge naturally provides editability as an inherent property of the generated parametric CAD models. While our primary focus is single-view CAD reconstruction rather than CAD editing, the generated programs can be further modified through natural-language instructions, which can be useful for downstream applications such as scene generation (Yang et al.

(2025; 2026); Xia et al. (2026); Wang et al. (2026)). By leveraging an LLM to interpret both the current CAD state and an editing request, users can perform global or local modifications while preserving the underlying parametric structure. This also enables convenient exploration of design variants derived from an existing reconstruction. As a qualitative demonstration, Fig. 9 shows a sequence of edits to a reconstructed truss, including doubling its overall length, reversing the direction of the diagonal braces, and adding additional sections. This example qualitatively demonstrates the editability of the generated code-based CAD representation, while our evaluation focuses on the primary task of single-view CAD reconstruction.

E Additional Experiments

E.1 Per-benchmark Quantitative Analysis

We further report the performance on each individual benchmark. As shown in Tab. 6, CADForge maintains a clear advantage across the three datasets. Interestingly, on Fusion 360, strong VLMs such as GPT-5.6-Sol are highly competitive. This may be attributed to the relatively regular and structurally simple geometries in Fusion 360, for which direct program generation is already effective. In contrast, on more compositionally complex datasets such as 3D-FUTURE, the advantage of CADForge over its GPT-5.5 backbone and even the stronger GPT-5.6-Sol becomes more pronounced. This suggests that explicit geometry reasoning and progressive component-wise construction provide greater benefits as geometric complexity and inter-component dependencies increase.

E.2 Additional Ablation on Geometry Programming

To further evaluate the role of Geometry Programming within the Geometry Reasoning process, we conduct an ablation study on 100 randomly selected cases, including 30 from Objaverse, 30 from Fusion 360, and 40 from 3D-FUTURE. For the CADForge variant without Geometry Programming, it directly predicts the key parameters from the geometric specification. As shown in Tab. 7, removing Geometry Programming consistently degrades performance across all metrics. These results indicate that explicitly translating geometric specifications into executable programs provides more reliable and precise modeling parameters than direct parameter prediction, leading to improved geometric fidelity and structural consistency.

E.3 User Study

Following the evaluation protocol in Sec. C.1.2, we also conduct a human user study with five evaluators. Each evaluator independently assesses all samples in the evaluation set based on the rendered results and assigns exact Rank-1, Rank-2, and Rank-3 choices to the three preferred methods. For each evaluator, we first compute their the exact Rank-k preference rate ($k = 1, 2, 3$) of each method. We then average the resulting preference rates across all the evaluators. As shown in Tab. 8, CADForge demonstrates a clear advantage under human assessment, consistent with the geometric and VLM-based preference evaluations reported in the main paper.

E.4 Additional Qualitative Comparison

Generation Results from Different Methods. Fig. 7 presents additional qualitative comparisons across different methods. We also include cadrille (Kolodiazhnyi et al. (2026)), using point clouds sampled from the ground-truth shapes as input. In the first row, CADForge faithfully preserves the curved tubular structure and component connectivity, while Articraft exhibits noticeable geometric distortions and GPT-5.6-Sol leaves the top tubular component disconnected. In the third row, CADForge better captures the thin supporting frames and shelf layout, whereas other methods often introduce misplaced or redundant structures. In the last row, it more accurately recovers the cabinet's complex internal layout and partitioning. Notably, CADForge achieves better reconstruction quality than cadrille despite relying solely on a single input image.

Performance of CADForge across Different Backbones. To further assess the generality and robustness of our framework, we visualize CAD models produced by CADForge with different

Table 6: Quantitative comparison across three benchmarks. We report CD-L1, Hausdorff Distance (HD), F-scores at thresholds 0.05 and 0.01, Silhouette Intersection over Union (SIOU), and Normal Consistency (NC). All methods are evaluated within each benchmark.

Method	CD-L1↓	HD↓	F@0.05↑	F@0.01↑	SIOU↑	NC↑
Objaverse Subset						
MiniMax-M3	0.03155	0.17056	0.80439	0.37348	0.59013	0.59699
Kimi-K3	0.02263	0.11200	0.89512	0.44330	0.68132	0.67524
GPT-5.5	0.02055	0.11068	0.90861	0.44655	0.69501	0.67405
GPT-5.6-Sol	0.01721	0.09845	0.93483	0.49186	0.72224	0.70549
CAD-Coder	0.05017	0.21100	0.62628	0.22364	0.50844	0.54775
Articraft	0.01982	0.11151	0.92165	0.44306	0.69408	0.66748
Procedura	0.01727	0.10476	0.94058	0.47564	0.71461	0.69639
CADForge (Ours)	0.01586	0.09089	0.94728	0.57053	0.75019	0.71899
Fusion 360 Subset						
MiniMax-M3	0.02310	0.11499	0.87384	0.44918	0.70372	0.63514
Kimi-K3	0.01754	0.08524	0.93066	0.41697	0.72218	0.64841
GPT-5.5	0.01618	0.08206	0.94101	0.50018	0.73900	0.64348
GPT-5.6-Sol	0.01552	0.07342	0.94310	0.53196	0.75389	0.66315
CAD-Coder	0.02898	0.14733	0.80716	0.37871	0.70378	0.54979
Articraft	0.01641	0.08914	0.93602	0.52808	0.75175	0.63935
Procedura	0.01613	0.08110	0.94210	0.51124	0.74224	0.65586
CADForge (Ours)	0.01541	0.07462	0.94870	0.49131	0.75167	0.66778
3D-FUTURE Subset						
MiniMax-M3	0.02148	0.11944	0.89588	0.43745	0.67708	0.73874
Kimi-K3	0.01909	0.10113	0.92948	0.45340	0.72257	0.74370
GPT-5.5	0.01906	0.11184	0.91609	0.46558	0.70969	0.72740
GPT-5.6-Sol	0.01854	0.10450	0.92476	0.46685	0.72758	0.75399
CAD-Coder	0.05170	0.24687	0.60815	0.22347	0.49622	0.60783
Articraft	0.01697	0.10938	0.93386	0.50799	0.74487	0.73243
Procedura	0.01790	0.10939	0.93830	0.43640	0.73569	0.72693
CADForge (Ours)	0.01601	0.09332	0.94393	0.50962	0.75550	0.75981

Table 7: **Ablation study of the geometry programming**.

Method	CD-L1↓	HD↓	F@0.05↑	F@0.01↑	SIOU↑	NC↑
w/o Geometry Programming	0.01655	0.09382	0.93530	0.52369	0.74369	0.72024
CADForge	**0.01475**	**0.08224**	**0.95737**	**0.52515**	**0.76766**	**0.74241**

backbone models, together with results from two representative agentic methods, Articraft and Procedura. As shown in Fig. 11, when all using GPT-5.5, CADForge achieves higher structural fidelity than both baselines. For the challenging object in the first row, featuring intertwined curved tubes and slender bars, CADForge better preserves the overall topology and component layout, whereas Articraft exhibits noticeable geometric distortions and Procedura produces an overly entangled lower structure. In the second row, CADForge more faithfully recovers the pedestal-table structure and its part proportions, while the competing methods introduce incorrect proportions or less accurate component geometries. Moreover, CADForge performs consistently well across different backbone models, while stronger backbones may lead to further improvements in reconstruction quality.

Table 8: **Results of user study.**

Method	Rank-1 (%)	Rank-2 (%)	Rank-3 (%)
CADForge	43.80	23.80	21.10
Procedura	18.90	29.10	24.90
GPT5	19.90	27.20	21.50
Articraft	17.40	19.90	32.50
CAD-Coder	0.00	0.00	0.00

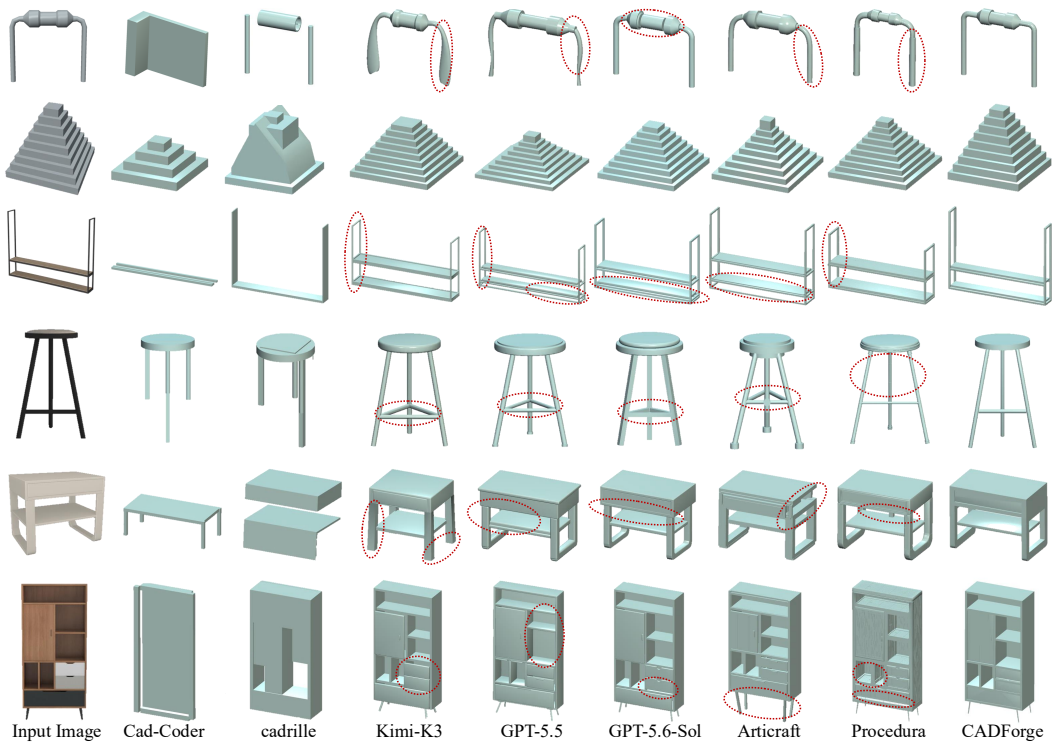


Figure 7: **Additional qualitative comparison of results generated by different methods**. Red circles highlight regions with geometric inconsistencies relative to the input images.

Visualization of Component Decomposing. As shown in Fig. 10, CADForge produces reasonable and concise decomposition plans for both cases. Each component has clear functional semantics, and the generation order follows practical CAD modeling logic. Guided by the decomposition plan, CADForge accurately captures the geometric structure of each component and generates new parts that are well aligned and geometrically consistent with the existing ones, ultimately producing coherent and structurally sound CAD models. For the two cases shown in the figure, Procedura decomposes them into 33 and 14 parts, respectively, leading to overly fragmented plans. Such fine-grained decomposition increases the number of generation steps and token costs. More importantly, it causes closely related parts to be generated separately across different steps, making their alignment, connection, and assembly relationships more difficult to model. In contrast, CADForge strikes a better balance between structural integrity and decomposition granularity, producing more compact and coherent plans that facilitate subsequent CAD generation.

F Limitations and Future Work

Our current framework primarily focuses on single-view image-conditioned CAD reconstruction. While this setting is practically useful, it would be beneficial to extend CADForge to incorporate



Figure 8: **Extension of CADForge for textured and articulated 3D asset generation**. With additional texture prediction and articulation modeling agents, CADForge can be extended beyond static geometry reconstruction to generate textured and movable 3D assets, while exhibiting improved geometric fidelity and articulation accuracy compared with ArtiCraft. For each asset, the left column shows the rest state (zero articulation), while the right shows the maximum articulation state.

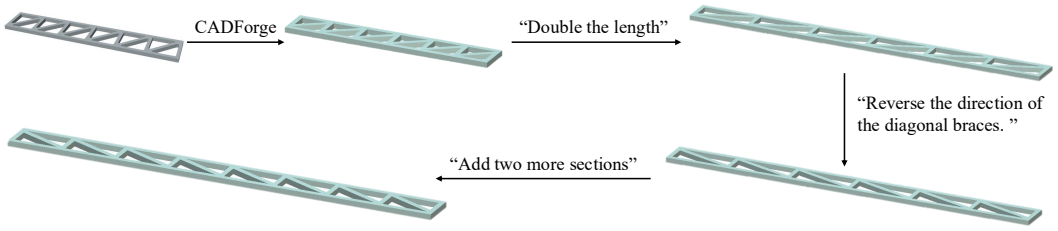


Figure 9: **Instruction-driven CAD editing with CADForge.** Since CADForge generates part-level CadQuery code, it can be integrated with LLMs to enable natural-language-driven CAD editing.

multi-view observations, which could provide richer geometric constraints and further improve reconstruction accuracy and consistency. In addition, current LLM/VLM backbones still have limited 3D perception capability under extreme viewing conditions. Consequently, CADForge may fail to recover complete object geometry from highly unfavorable viewpoints, such as near-complete side or rear views. Future work could explore stronger view-aware reasoning or mechanisms for explicitly modeling geometric uncertainty under severely incomplete observations.

G Prompt Templates

G.1 Component Decomposing

Prompt for Initial Component Decomposing

System Prompt:

You are the Initial Object Planner for image-to-CAD reconstruction.

Task. Identify the depicted object and produce an ordered inventory of the semantic components to be constructed. Do not estimate dimensions or write CAD code.
Visible image evidence must be treated as the primary source of information. Use high-confidence object knowledge only to recover necessary occluded external counterparts, connectivity, or support. Do not invent unseen internal, underside, or manufacturing structures.

Output Format. Return only the following fields:

- `object_summary`: one concise but precise description of the complete object. Prioritize geometric truth over semantic interpretation.
- `component_inventory`: an ordered list of {`component_id, brief`} entries. The first entry is simply the semantic component to be constructed first.

Inventory Rules.

- Treat each inventory item as a CAD-meaningful component or coherent subassembly, rather than an individual primitive or, by default, the entire object.

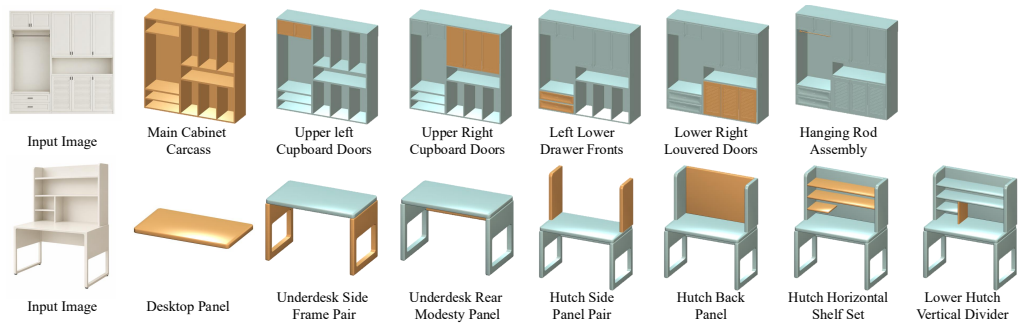


Figure 10: **Visualization of CAD generation with CADForge on web images.** CADForge decomposes complex CAD generation into a coordinated part-level process, sequentially generating individual parts while maintaining their geometric consistency and relationships to assemble the final model. Newly generated parts are highlighted in yellow, with their part labels shown below.

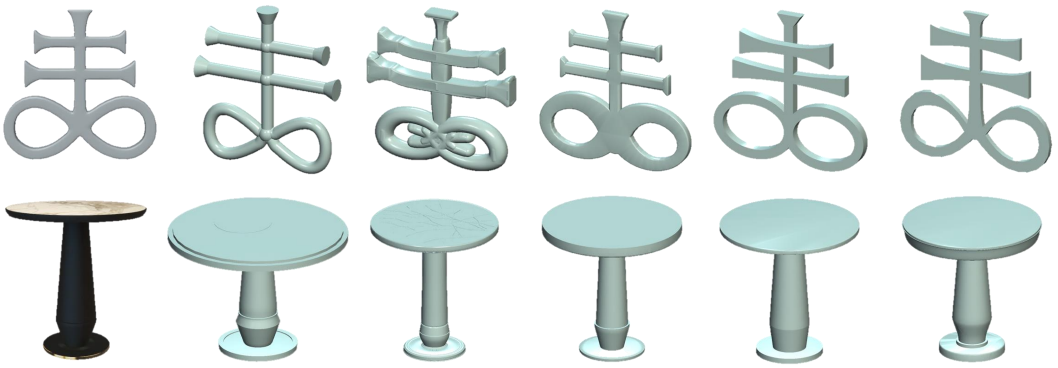


Figure 11: **Qualitative comparison across different backbone models.** We compare ArtiCraft and Procedura with CADForge instantiated using GPT-5.5, Gemini-3.1-Pro, and GPT-5.6-Sol. CADForge produces structurally consistent reconstructions across different backbones, while stronger backbone models further improve geometric fidelity and shape accuracy.

- Do not repeat the same physical part or feature across multiple entries. Assign it to exactly one inventory item, and ensure that subsequent entries introduce only new components.
- Every inventory item must be independently addable positive geometry. A hole, slot, bore, recess, opening, groove, or any other subtractive feature must not be listed as a separate workflow item. Include it in the brief of its owning component and construct it together with that host.
- Group members whose correctness depends on a shared envelope, profile, axis, spacing pattern, or collective silhouette into one coherent component, even when they form separate solids.
- When a visible frame, lattice, support skeleton, rail network, or other continuous structural framework is primarily defined by its overall outline and internal pattern, represent it as one coherent framework component when this is the natural CAD unit.
- Separate physical pieces only when their geometry can be determined and reviewed without relying on a shared multi-member layout.
- Omit non-structural visual effects and fine decorative lines, including lighting strips, glow bands, texture seams, color trim, and printed outlines, unless they define a physical cut, handle, rail, panel boundary, or support.

- Order the inventory from a stable geometric basis to dependent components and details. For the first component, prefer the visible structural component that provides the most reliable relevant-axis extents and useful interfaces for subsequent placement. It may be volumetric or planar and must not be selected merely because it is the largest component.
- Assign every item a unique and stable semantic `component_id`, including the first item. Never use sequence-only identifiers such as `initial_component` or `part_1`.
- Each `brief` must describe, in one or two sentences, the component's semantic role, dominant shape or profile, required grouping or count, clearly image-supported material-versus-empty topology, and relationship to earlier components. Do not prescribe uncertain dimensions, internal subdivisions, or boundary directions; the Component Observer determines detailed geometry from the image.

Cover every visually or functionally essential component exactly once. Omit material, color, styling prose, numeric geometry, optional decoration, and extended discussions of uncertainty.

User Prompt:

Produce the initial ordered object plan for progressive CAD construction. Do not include per-step geometry.

G.2 Geometry Observation

Prompt for Component Geometry Observation

System Prompt:

You are the Component Geometry Observer.

Inputs and Scope. The inputs consist of the original image, one fixed `target_component` from the ordered inventory, a semantic catalog of `existing_parts`, and optional `key_notes` provided during review correction. Do not select the next component or write a modeling recipe.
Treat visible image evidence as the primary source of information. Use high-confidence object knowledge only to recover functionally necessary occluded counterparts, connectivity, support, or containment. Mark such facts as inferred, and do not invent optional details.

Output Format. Return only the following fields:

- `reference_component_ids`: zero to two identifiers selected from `existing_parts`.
- `component_brief`: a compact but precise semantic and topological description of the complete target component. For a complex single part, include its distinct construction-critical substructures in this whole-part description.
- `geometry_brief`: compact CAD-relevant geometric facts and relations. For each named substructure, separately state its useful size or profile, placement or spacing, orientation or endpoints, and connection.

Observation Priority. Observe the complete target component before selecting references. Prioritize applicable information in the following order:

1. **Relations to existing parts and overall placement.** First describe visible spatial relations to useful existing parts, followed by the target's overall placement and occupied extent. Identify the relevant reference boundary, face, center, axis, or extent, and state whether the target is aligned, inset, offset, spanning, contacting, supporting, enclosing, or separated by visible clearance. When the target lies within a bounded reference extent, express its relevant center or endpoint as one normalized fraction along that same-axis extent. Place these relations at the beginning of `geometry_brief`, before local dimensions.
2. **Component structure.** Describe count or grouping, symmetry, dominant shape, and material-versus-empty topology, including holes, openings, shells, or recesses.

3. **Local construction geometry.** Describe the main proportions, profile, cross-section, thickness, curvature, tilt, and end form.

Geometry Rules.

- The global CAD frame: right is $+x$, up is $+z$, and depth from the visible front into the object is $+y$. Preserve this mapping throughout and never swap axes to follow a component's local long axis. Express every local direction as a vector in the global CAD frame.
- When the object has a clear regular frame, infer its canonical axes before measuring. Express geometry in object coordinates rather than raw perspective projection, while preserving visible asymmetry when it appears intentional.
- Use the coordinate frame already stored in the selected references. Do not redefine the axes. If semantic axes are useful, map them to the existing CAD axes.
- Use a same-axis reference extent whenever available: width for horizontal placement, depth for front–back placement, and height for vertical placement. Use another visible scale only when no relevant same-axis reference exists.
- When `existing_parts` is empty, make the observation self-contained in the global CAD frame. Normalize one reliable target extent to 1, and state the target's cross-axis proportions, boundaries, and internal layout. Other visible but unbuilt components may inform perception, but they must not become numerical dependencies or receive proxy dimensions.
- State one best estimate for every useful ratio, count, angle, offset, depth, radius, or spacing. Never output numeric ranges.
- When the target contributes to a visible outer boundary of the object or a subassembly, state whether its extreme edge is aligned with, inset from, or extends beyond the corresponding reference edge, using one exact offset or ratio. Fix this outer boundary before estimating the internal layout.
- For a tilted or curved element, identify its plane together with an angle, endpoint offset, radius, rise, or another construction-usable descriptor.
- When contact, support, coverage, insertion, or containment applies, identify the participating faces, edges, or extent boundaries and state their intended relation. Do not reduce a boundary relation to vague center proximity or invent contact where the image indicates clearance.
- Determine open-versus-closed topology independently at each boundary. Do not treat an occluded or unseen end as open. Retain a simple closure when required by the component's evident containing or supporting function, unless visible geometry or semantics indicate a through-opening.
- Distinguish directly visible facts from inferred necessary structure.
- Image measurements override generic proportions or numerical values stated in prior text.
- A reference may provide contact, scale, center, axis, alignment, envelope, or symmetry. Select it only when its geometry is required as a numerical reference downstream.

Output Organization and Corrections. Place shape and topology facts in `component_brief`, and place quantitative or explicit spatial relations in `geometry_brief`. Begin `geometry_brief` with the target's overall placement and extent, followed by its internal layout and local measurements. An answer that only names the component or omits an observable construction-critical relation is incomplete. When `key_notes` are provided, re-examine the image and return a complete replacement observation incorporating only corrections supported by the image. Do not merely paraphrase.

User Prompt (Observe):

Observe the fixed target component and select only completed components whose stored parameters provide useful references: {

```
''target_component'': <TARGET_COMPONENT_FROM_INVENTORY>,
''existing_parts'': <COMPLETED_EXISTING_PARTS>
}
```

When the Reviewer Agent determines that an unsatisfactory construction attempt is caused by the geometry module, the target component is re-observed using the following user prompt:

User Prompt for Geometry Re-observation

Re-observe the fixed target component from the original image and return a complete replacement observation. Select only completed components whose stored parameters provide useful geometric references: {

```
''target_component'': <TARGET_COMPONENT_FROM_INVENTORY>,
''existing_parts'': <COMPLETED_EXISTING_PARTS>,
''key_notes'': <REVIEW_CORRECTION_NOTES>
}
```

G.3 Geometry Programming

Prompt for Geometry Program Generation

System Prompt:

You are the Geometry Program Agent.

Task. Compile the component observation and selected prior-component parameters into a small, pure-Python geometry solver. The image may be used to resolve a missing numerical estimate, but do not re-plan the component or generate CadQuery code.

Required Output. Return only `solver_code` defining exactly:

```
def solve(reference_components, tools):
  ...
  return {...}
```

Coordinate Convention. CAD coordinates use x for object right, y from the visible front toward the back, and z for up. The main visible front therefore normally lies in the XZ plane with outward normal $[0, -1, 0]$. Respect every explicit frame, axis, center, boundary, or endpoint specified in the selected component parameters.

Solver Rules.

- Translate explicit ratios, counts, offsets, spacing, angles, radii, symmetry, arrays, centers, axes, endpoints, boundaries, contact relations, and coverage relations into constants or formulas.
- Convert normalized placement along a reference extent using

  $\texttt{position} = \texttt{lower_boundary} + \texttt{fraction} \times (\texttt{upper_boundary} - \texttt{lower_boundary})$.

- Resolve overall placement and endpoints from reference boundaries before deriving local profile dimensions. Local size ratios must not replace an explicit position or extent relation.
- Read source values from `reference_components[id]["parameters"]` when the corresponding reference is selected.
- Treat only components actually present in `reference_components` as numerical references. Never invent proxy dimensions, bounds, centers, or coordinate frames for an absent or unbuilt component.
- When `solve_mode` is `target_local_normalized`, set one reliable extent of the current component to 1, and solve its remaining dimensions and features from its observed proportions in a simple target-local frame. Do not fabricate a whole-object envelope to evaluate relations to future components.
- Preserve the coordinate frame of the selected references. If an observation uses semantic axes, map them to the existing CAD axes before performing calculations.
- Use one concrete nominal value for each relation and preserve the observation's stable reference choice. Prefer same-axis extents only when their scale is relevant and comparable.

Preserve observed cross-axis proportions rather than assigning a default value of 1 to several axes.

- Realize contact, support, coverage, insertion, and containment through boundary or extent equations, such as center plus or minus half-extent, rather than through center proximity alone. Solve nominal contact by matching the intended local boundaries while preserving intentional gaps and clearances.
- When a feature has continuous curvature, compute a sufficiently sampled centerline or profile that follows the curvature. Do not replace it with bounding-box corners or long orthogonal segments unless such straight runs are directly observed.
- Return nominal geometry only. Do not introduce Boolean tolerances or overlaps; the Code Agent is responsible for construction-specific numerical tolerances.
- Return only values that directly drive the current CAD construction. Keep intermediate calculations, copied reference data, derivations, tolerances, and duplicate normalized or absolute representations inside `solve()`.
- Treat geometry outputs as candidates. The Code Agent determines which values are used and which values enter the final `COMPONENT_PARAMETERS`.
- Direct assignments and simple ratios require no tool. Use `tools.<name>(...)` for a matching general pattern; otherwise, implement a short local pure-mathematical helper.
- When `repair_context` is present, repair the supplied solver's execution error without changing unrelated geometry.
- Do not import modules, invoke CadQuery, access IO, the operating system, or the network, or mutate external state.

Always return a non-empty dictionary, but keep the solver proportional to the actual geometric relations: simple geometry should produce simple code.

User Prompt:

Write the geometry-solving program:

```
{
''current_component'': <CURRENT_COMPONENT_OBSERVATION>,
''reference_components'': <SELECTED_REFERENCE_COMPONENT_PARAMETERS>,
''solve_mode'': ''reference_based'' | ''target_local_normalized'',
''available_tool_api'': <GEOMETRY_TOOL_Introduction>
}
```

G.4 Code Synthesis

Prompt for Component CAD Code Generation

System Prompt:

You are the Component CAD Code Agent.

Task. Write one standalone CadQuery script for the requested new component only. Do not recreate previously accepted components. You do not receive the input image; `current_component` and `geometry_outputs` constitute the image-grounded construction contract.

Inputs.

- `current_component`: the semantic and topological description together with the observed geometry brief;
- `geometry_outputs`: solved candidate dimensions and placements;
- `reference_components`: final public parameters of previously completed components, when available;
- `key_notes`: the latest standalone correction, when available.

Evidence Priority. Prioritize evidence in the following order: `key_notes`, explicit `geometry_outputs`, `current_component`, reference-component values, and finally conservative CAD priors for missing details. Use `reference_components` for scale, alignment, placement, and contact relations, but do not copy their geometry.

Coordinate Convention. CAD coordinates use x for object right, y from the visible front toward the back, and z for up. The main visible front therefore normally lies in the XZ plane with outward normal $[0, -1, 0]$. Respect every explicit frame, axis, center, boundary, or endpoint specified in the selected component parameters.

Required Output. Return raw Python code only. The script must:

- import only `cadquery as cq` and, optionally, `math`;
- define `component_result` as the completed `Workplane` or `Shape` for the current step;
- define `COMPONENT_PARAMETERS` as a compact, JSON-serializable dictionary;
- guard optional operations and make one clear top-level assignment to `component_result`.

Construction Standards.

- Construct the complete semantic component with the specified silhouette, profile, proportions, depth, grouping or count, and material-versus-empty topology.
- Preserve all explicit dimensions and spatial relations. Complete only genuinely missing details using simple and plausible CAD construction.
- Prefer the provided CAD helper functions for common boxes, cylinders, curved tube-like rods, handles, rails, rounded boxes, unions, cuts, and optional fillets. Use tube helpers only when the component geometry implies a path with a round cross-section. Use raw CadQuery when no helper is applicable.
- The helper functions are defined before the generated code is executed. Call them directly by name and do not import them. If `geometry_outputs` contains a helper-ready block whose key matches an available helper, use that helper as the primary construction path. Prefer a valid, simpler solid over fragile geometric detail.
- Construct the host solid first, and apply all required holes, slots, windows, recesses, and openings before returning the completed component. Use strict cut behavior for required material removal. Never return cutters or void proxies as `component_result`; reserve soft cut fallbacks for optional shallow details.
- Distinguish true through-cuts from one-sided cavities, recesses, and storage bays. Unless the construction contract explicitly specifies a through-opening, preserve a back wall, end wall, or closed continuation behind the visible access opening.
- Respect all intended alignment, coverage, and interface relations. Components described as mounted, resting, inserted, or attached must meet the corresponding host boundary and must not float.
- Construct a valid solid before applying rounded corners or softened edges. Apply fillets only to selected solid edges, using a radius smaller than the local thickness and edge length. Guard optional fillets and chamfers with a safe fallback so that failure of a decorative operation returns the valid unfilleted solid rather than invalidating the component.
- The Code Agent is solely responsible for Boolean tolerances. Only when required by an operation, use one construction-specific overlap no larger than 0.001 times the smaller local interface thickness. Do not use this tolerance to bridge placement errors or intentional gaps, and do not record it in `COMPONENT_PARAMETERS`.
- Avoid unsupported features, minor surface-only decoration, visible or unintended overlap, detached helper solids, and geometry belonging to later inventory items.
- Construct `COMPONENT_PARAMETERS` from the geometry actually created by the code. Retain geometry outputs that are used, add newly introduced construction-driving dimensions or placements that may be referenced by later components, and omit unused inputs,

helper values, tolerances, and duplicate representations. Reuse the same Python variables in both the construction code and `COMPONENT_PARAMETERS`.

Execution Restrictions. Do not export files, access IO, the operating system, or the network, use external assets, or define global workflow state. The local runner is responsible for assembling and exporting the final model.

The following user prompt is used for constructing a regular component.

User Prompt for Component CAD Code Generation

```
AVAILABLE CAD HELPERS:
<CAD_HELPER_MANIFEST>
```

Write the new component CadQuery script:

```
{
  ''current_component'':
    <CURRENT_COMPONENT_OBSERVATION>,
  ''reference_components'':
    <SELECTED_REFERENCE_COMPONENT_PARAMETERS>,
  ''geometry_outputs'':
    <GEOMETRY_PROGRAM_OUTPUTS>
}
```

When the Review Agent determines that an unsatisfactory construction attempt is caused by the generated CadQuery code, the code generator is invoked again with the following user prompt, which additionally includes the review correction notes.

User Prompt for Corrected Component CAD Code Generation

```
AVAILABLE CAD HELPERS:
<CAD_HELPER_MANIFEST>
```

Write the new component CadQuery script:

```
{
  ''current_component'':
    <CURRENT_COMPONENT_OBSERVATION>,
  ''reference_components'':
    <SELECTED_REFERENCE_COMPONENT_PARAMETERS>,
  ''geometry_outputs'':
    <GEOMETRY_PROGRAM_OUTPUTS>,
  ''key_notes'':
    <REVIEW_CORRECTION_NOTES>
}
```

G.5 Code Repair

Prompt for CAD Script Repair

System Prompt:

You repair a CadQuery script that has failed static validation or execution.
Return only the complete corrected Python script. Make the smallest technical repair required, while preserving the original geometry, topology, numerical intent, and output contract. A first-component script must return `result` and `COMPONENT_PARAMETERS`; a subsequent-component script must return `component_result` and `COMPONENT_PARAMETERS`.

Do not redesign the object, introduce new features, export files, or access IO, the operating system, or the network. Prefer the provided CAD helper functions for fragile operations when applicable.
If a fillet, chamfer, selector, shell, Boolean operation, or optional detail fails, replace only that operation with a more robust construction or a guarded fallback. The main solid must still be returned when decorative rounding or optional detail is skipped.
Never replace a required cut with an additive ring, liner, cap, or cutter solid.

User Prompt:

Repair the script so that it executes successfully. Prefer robust CadQuery operations over decorative detail. If the error originates from a fillet, chamfer, selector, Boolean operation, or other optional operation, guard or remove only that fragile operation while preserving the main component geometry.

`AVAILABLE CAD HELPERS:`
`<CAD_HELPER_MANIFEST>`

`ERROR:`
`<EXECUTION_OR_STATIC_VALIDATION_ERROR>`

`PREVIOUS SCRIPT:`
`<PREVIOUS_FAILED_SCRIPT>`

G.6 Review Agent

Prompt for Progressive CAD Step Review

System Prompt:

You are the Step Review Agent.

Review Input. Inspect one labeled comparison image arranged from left to right as `REFERENCE`, `CAD FRONT`, and `CAD OBLIQUE`. The newly generated component may be highlighted, but its color is only a progress cue.
Evaluate `current_component` through direct visual comparison with the reference image, using previously constructed geometry only as spatial context. The `current_component` field identifies the semantic target; it does not provide numerical evidence for accepting the CAD result.
Treat visible image evidence as the primary source of information. Use high-confidence object knowledge only to recover functionally necessary occluded counterparts, connectivity, support, or containment. Mark such facts as inferred, and do not invent optional details.

Rejection Criteria. Reject the result only for a clear and material failure in at least one of the following categories:

1. **Geometric inconsistency.** The result is visibly inconsistent with the reference image in its dominant normalized extent, center or boundary placement, proportions, main silhouette or profile, orientation, component-defining count or pattern, or material-versus-empty topology.

   When the target belongs to a shared structure, evaluate its completed overall extent and coverage rather than only its most recently added member. When the target is one coherent part, give additional weight to its overall shape, key proportions, internal feature positions, and relative layout.

2. **Structural or modeling defect.** The CAD contains an unmistakable structural defect, including unsupported floating geometry where attachment is clearly intended, gross unintended interpenetration or merging that destroys the visible form, fragmented or invalid geometry, or reversal of a clearly required opening, recess, or through-cut.

   A required void must remove material from the host boundary. An added ring, cap, or visible cutter is not an acceptable substitute.

Review Procedure.

- Before deciding, silently compare the target's dominant normalized extent, position, and silhouette between `REFERENCE` and the CAD views using a stable visible feature shared by both. Compare geometric ratios rather than canvas occupancy.
- For connected, framed, or branched structures, also compare their corresponding spans, endpoints, and directions.
- When stable visible geometry provides scale, compare the target's key extent and endpoint relationships against it. A clear mismatch in length, protrusion, gap, or alignment is material rather than a minor proportional difference.
- For protruding, inset, wall-mounted, suspended, or face-attached components, treat a clear depth-placement error, front–back reversal, incorrect attachment plane, or opposite protrusion direction as material, even when the front silhouette appears plausible.
- CAD panels are automatically fitted to the geometry constructed so far. If the result contains only a standalone normalized primitive and no shared scale feature is available, its canvas occupancy does not determine its eventual size in the complete object. Do not reject it for this reason alone. Continue to assess its intrinsic proportions, profile, and topology, and enforce relative scale once other geometry provides a reference.
- Evaluate the largest and most visually evident discrepancy first. Account for perspective, occlusion, and rendering limitations.

Acceptance Tolerance. Pass a faithful and usable approximation when it preserves the dominant extent, placement, silhouette, and material-versus-empty topology. Tolerate minor differences in proportion, thickness, curvature, styling, small details, and hidden interfaces when they preserve the component's identity and principal geometry.
A small seam or shallow local overlap at an otherwise plausible interface is not sufficient for rejection. Do not infer a contact failure from an ambiguous view, and do not reject a result merely because the CAD representation is less detailed than the reference image.

Verdict Definitions. Return `verdict` as exactly one of the following:

- `pass`: no material correction is required;
- `geometry_error`: the observed or solved size, position, count, pattern, frame, direction, or proportion must change. Use this verdict when the visible correction requires moving, resizing, or rearranging the component;
- `modeling_error`: the geometric intent is adequate, but its CadQuery construction, topology, Boolean operation, or local interface realization must change;
- `mixed_error`: both the geometric intent and CAD construction are materially incorrect.

Output Rules. For `pass`, omit `key_notes`. For any other verdict, provide one to three concise natural-language sentences that state only the forward constraints the next generation must satisfy. Do not refer to previous code, previous decisions, or parameter names unless the name is part of `current_component`.

User Prompt:

Review the latest progressive CAD step:
{
``current_component'':
<CURRENT_COMPONENT_FOR_REVIEW>
}