CADReview: Automatically Reviewing CAD Programs with Error Detection and Correction

Jiali Chen[1,2]*, Xusen Hei[1,2]*, Hongfei Liu[1,2], Yuancheng Wei[1,2], Zikun Deng[1,2], Jiayuan Xie[3], Yi Cai[2,1]†, Qing Li[3]

[1]School of Software Engineering, South China University of Technology,
[2]Key Laboratory of Big Data and Intelligent Robot Ministry of Education,
[3]The Hong Kong Polytechnic University

segarychen@mail.scut.edu.cn {zkdeng, ycai}@scut.edu.cn
{jiayuan.xie, qing-prof.li}@polyu.edu.hk

†**Correspondence:** ycai@scut.edu.cn

Abstract

Computer-aided design (CAD) is crucial in prototyping 3D objects through geometric instructions (*i.e.*, CAD programs). In practical design workflows, designers often engage in time-consuming reviews and refinements of these prototypes by comparing them with reference images. To bridge this gap, we introduce the CAD review task to automatically detect and correct potential errors, ensuring consistency between the constructed 3D objects and reference images. However, recent advanced multimodal large language models (MLLMs) struggle to recognize multiple geometric components and perform spatial geometric operations within the CAD program, leading to inaccurate reviews. In this paper, we propose the CAD program repairer (ReCAD) framework to effectively detect program errors and provide helpful feedback on error correction. Additionally, we create a dataset, *CADReview*, consisting of over 20*K* program-image pairs, with diverse errors for the CAD review task. Extensive experiments demonstrate that our ReCAD significantly outperforms existing MLLMs, which shows great potential in design applications[1].

1 Introduction

Computer-aided design (CAD) plays a critical role in industrial design and manufacturing, serving as the foundation for product prototyping (Haque et al., 2022; Jones et al., 2023). It represents 3D objects through sequences of geometric instructions, commonly referred to as *CAD programs*, which define editable geometric components and operations. Despite the emergence of various 3D modeling software (*e.g.*, AutoCAD, SketchUp, Rhino and FreeCAD), the design workflow persists as a technically challenging process. It is time-consuming and requires specialized expertise from designers.

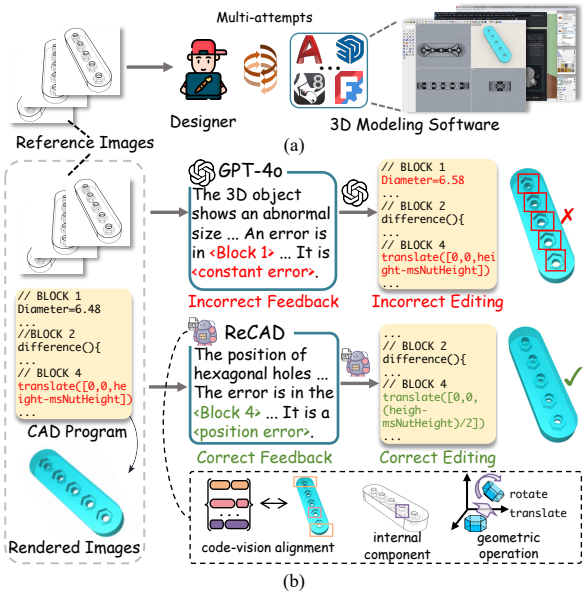


Figure 1: Overview of different ways (*i.e.*, designer, GPT-4o and our ReCAD) to CAD review. The green and red denote correct and incorrect prediction, respectively.

Towards this end, several generative models have emerged for 3D object generation through CAD programs (Haque et al., 2022; Xu et al., 2022; Khan et al., 2024a). However, the generated 3D object may contain errors, and the practical requirements in industrial design extend far beyond mere initial 3D generation. In realistic design workflows, designers conduct meticulous reviews and refine the 3D object using modeling software (Chang, 2015), a process that requires multiple attempts to ensure consistency with the design drawing (*i.e.*, reference image), as shown in Fig. 1 (a). In contrast, AI models can directly analyze and edit CAD programs for this process. Motivated by this intuition, we propose the CAD review task to automatically detect and correct program errors, addressing discrepancies between the 3D object and reference images. Following previous works (Chae et al., 2024; Tang et al., 2024) on code review, our task

*Equal Contribution.

[1]Our dataset and code are released at the project page: https://cgl-pro.github.io/cadreview

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 9909–9927
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

aims to generate helpful feedback for program error detection and then utilize this feedback to edit the erroneous code, as illustrated in Fig. 1 (b).

Currently, advanced multimodal large language models (MLLMs), *e.g.*, GPT-4o (OpenAI, 2023), have shown remarkable performance across vision-language tasks (Yue et al., 2024; Chen et al., 2024a; Li et al., 2024c; Yuan et al., 2023, 2025). Nevertheless, they struggle to integrate CAD programs with reference images for the CAD review task. As illustrated in Fig. 1 (b), GPT-4o fails to detect program errors, primarily due to its limited ability to align code with visual information. Therefore, addressing the CAD review task requires overcoming multiple challenges. **First**, 3D objects typically consist of multiple components, each represented by a specific code block within the program. Moreover, these code blocks may contain intricate program structures (*e.g.*, subroutines and control flows), which recursively organize primitives (*e.g.*, cuboid and ellipsoid) into geometric components. The model demands correlating these components with corresponding code blocks. **Second**, 3D objects may contain hidden internal components that hinder visual inspection for humans and models, *e.g.*, internal hexagonal holes shown in Fig. 1 (b). It requires programmatic analysis of the CAD programs for error detection. **Third**, in addition to error detection, precise correction requires mapping geometric operations to corresponding code modifications (*e.g.*, rotation and translation). As shown in Fig. 1 (b), it needs to modify the component's offset along the z-axis.

To tackle the above challenges, we propose CAD Program Repairer (ReCAD), a MLLM-based framework, consisting of a feedback generator and code editor. Specifically, the feedback generator first aligns code with geometric components and then detects program errors for accurate feedback generation. Next, the code editor leverages this feedback to perform geometric operations by editing code for 3D object reconstruction. Moreover, we create *CADReview*, a benchmark dataset for the CAD review task, consisting of over 20K reference images, CAD programs, and annotated feedback. It contains 3D objects, each averaging over 8 geometric components, meeting the realistic industrial design requirements.

Our contributions are summarized as follows: **(i)** Motivated by the realistic design workflows, we introduce the CAD review task, which aims to detect and correct CAD program errors based on the reference images. **(ii)** We develop ReCAD, a MLLM-based framework that generates feedback to guide program error correction. Additionally, we create *CADReview* dataset as a new benchmark for the CAD review task. **(iii)** Experimental results demonstrate that our ReCAD achieves significant performance gains over existing MLLMs.

2 Related Work

CAD Program. Designing CAD programs is inherently challenging, as each code block on the final geometric output is often non-intuitive and difficult to foresee. Existing research (Haque et al., 2022; Xu et al., 2022; Slim and Elhoseiny, 2024) primarily focuses on using AI models to generate CAD programs. Specifically, DeepCAD (Haque et al., 2022) and SkexGen (Xu et al., 2022) are 3D generative models that represent shapes with CAD programs, offering a novel perspective on 3D representation. Text2CAD (Khan et al., 2024b) generates CAD programs based on textual modeling instructions, with diverse levels ranging from beginner to expert. Recent studies (Wu et al., 2024; Wang et al., 2024b) have explored using multimodal large language models (MLLMs) for CAD program generation. Specifically, CAD-GPT (Wang et al., 2024b) generates CAD programs based on multimodal input, which maps 3D spatial coordinates to a 1D linguistic feature space through tokenization method. Despite advancements, the generated CAD programs still exhibit discrepancies from the intended design. Since CAD designers spend considerable time reviewing 3D objects against design drawings, we introduce the CAD review task for automatic program error detection and correction. Furthermore, we also propose the *CADReview* dataset, which includes diverse geometric components and complex structures for review.

Code Edit. Large language models (LLMs) have achieved impressive results in code editing on generic programming languages (*e.g.*, C++ and Python). A common approach for code editing involves utilizing execution feedback from compilers or test cases (Zhang et al., 2023; Chen et al., 2024b; Gou et al., 2024) to correct code. Recently, natural language feedback has been utilized for code editing due to its interpretability. Specifically, Chae et al. (2024) design a reward function that reflects the helpfulness of feedback. CodeAgent (Tang et al., 2024) is an autonomous multi-agent system, which simulates collaboration among roles

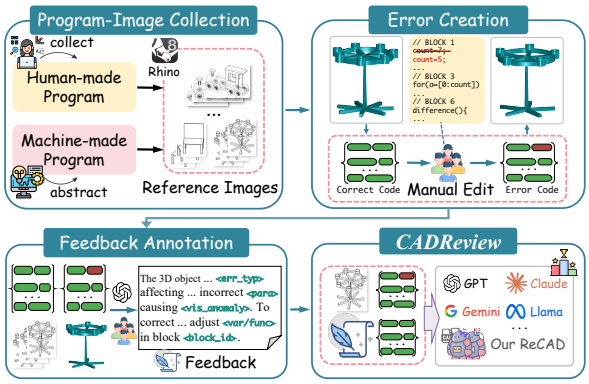


Figure 2: Construction workflow of our *CADReview* dataset, including program-drawing pairs collection, error creation and feedback annotation.

in practical software development to perform code editing. However, CAD programs fundamentally differ from generic programs, as their code blocks represent geometric components, and the editing process relies on design drawings. To the best of our knowledge, we are the first to introduce the task of CAD code editing, termed CAD review.

3 Building *CADReview* Dataset

While a few datasets of CAD programs have been recently proposed (Haque et al., 2022; Yuan et al., 2024), they do not include code segments that deviate from the design drawing (*i.e.*, reference image), which are critical for designers to rectify. Hence, we introduce *CADReview* dataset as a new benchmark for the proposed CAD review task. In total, we split 17,334, 2,000 and 1,615 data samples for training, validation and testing respectively. Our dataset contains CAD programs with potential errors and corresponding correct reference images. In the following sections, we describe the construction process, as shown in Fig. 2. More details are provided in Appendix A.

3.1 Collecting Program-Image Pairs

We collect two types of CAD programs (*i.e.*, human-made and machine-made programs), and obtain the corresponding design drawings as reference images. Human-made programs involve complex geometric instructions, *e.g.*, control flow, while machine-made programs are constructed from simple geometric primitives. It enables a comprehensive evaluation of program structures and geometric complexity. We choose OpenSCAD (Wikipedia, 2025a), an open-source CAD program language for its convenience in data collection.

Human-made Program. The human-made programs are created by experienced designers. We collect and filter 1.5*K* OpenSCAD programs from online design communities, with diverse 3D object categories, as shown in Fig. 7 of Appendix A.1. Specifically, we first render the corresponding 3D objects for each collected program. Considering that designers often divide 3D objects into distinct geometric components for review, we parse the CAD program into multiple code blocks as different components. To achieve this, we traverse the abstract syntax tree of the program from the top down. During traversal, we treat the irreducible parts, *i.e.*, macros, modules, control flows (*e.g.*, nested loops and conditional statements), boolean operations (*e.g.*, difference, union, intersection, *etc*), and geometric primitives (*e.g.*, cuboid, ellipsoid, *etc*) as independent code blocks. For simplicity, we comment block IDs before the corresponding code blocks and treat initial macros as the first block.

Machine-made Program. Previous studies (Jones et al., 2023; Yuan et al., 2024) have shown that 3D objects can be automatically abstracted as compositions of simple geometric primitives. Building on this insight, we choose three basic object categories (*i.e.*, chair, table and storage) from PartNet (Mo et al., 2019) and convert them into OpenSCAD programs with cuboid, ellipsoid and cylinder abstraction. Specifically, we use cuboids for the initial abstraction and then randomly replace some cuboids with ellipsoids or cylinders. After abstraction, we manually remove low-quality samples with excessive overlap of primitives. Similar to human-made programs, we also separate code blocks and assign corresponding block IDs. As a result, we obtain a total of 1.6*K* machine-made programs.

Reference Image. For each CAD program, we perform multiview rendering to obtain three representative design drawings of the 3D object as reference images. Specifically, we use the modeling software (*i.e.*, Rhino (Wikipedia, 2025b)) for rendering, as shown in Fig. 2. The viewpoints are manually sampled from a hemisphere around the object. In particular, we utilize a perspective view to render objects considering their potential internal components.

3.2 Creating Errors on CAD Programs

We manually modify programs to create errors, resulting in anomalous 3D objects that exhibit discrepancies with the reference images. Our dataset

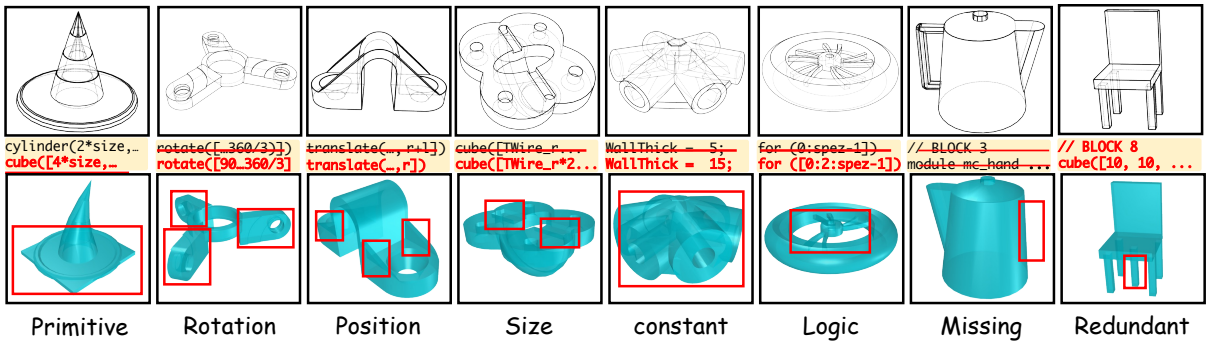


Figure 3: Examples of 8 types of CAD program error from our *CADReview* dataset. **Top row**: The reference images from one sampled viewpoint. **Second row**: Error creation on CAD programs, showing the edited code segments. **Bottom row**: The rendered 3D objects by erroneous programs.

includes 8 error scenarios that are relevant to real-world CAD review applications, as shown in Fig. 3. *(i) Primitive error* refers to the use of an incorrect geometric primitive. For instance, substituting a cube with a cylinder or a sphere leads to a misrepresentation of the intended design. *(ii) Rotation error* are created by applying a rotational transformation to the component of the 3D object. *(iii) Position error* pertains to deviations in the 3D coordinates of the components from their intended positions in the design. *(iv) Size error* is the discrepancy in the scale of the geometric component, similar to scenarios where parts of the object appear broken. *(v) Constant error* represents the errors in the initial macros or constants. These incorrect values or invalid assignments disrupt the intended 3D object generation. *(vi) Logic error* occurs in control flow statements, such as logical conditionals and loops, that cause unintended program execution paths. *(vii) Missing block* remove one code block, which results in incomplete 3D object construction or missing components. *(viii) Redundant block* involves the unnecessary code blocks that do not contribute to the intended 3D object.

We apply these errors to both human-made and machine-made CAD programs. Specifically, we develop a web interface to manually create errors on the CAD programs. Each annotator is assigned to create one error per annotation. A sourced program can be converted into multiple erroneous programs, where errors may appear in different code blocks. Additionally, normal samples with correct programs are also included in our dataset.

3.3 Annotating Feedback on CAD Programs

Inspired by previous studies (Chae et al., 2024) leveraging natural language feedback for code editing, we use GPT-4o (OpenAI, 2023) to annotate feedback for CAD programs. Our feedback focuses on evaluating the consistency of the CAD program with the reference images. The feedback provides a textual description of the visual anomalies, erroneous code blocks, and types of program errors. Specifically, for erroneous CAD programs, we first prompt GPT-4o to compare the reference images with the rendered output image, highlighting the differences and providing descriptions of the visual anomalies. Then, GPT-4o needs to identify the block IDs of the incorrect code and describe the errors based on the original correct code. After that, we manually recheck each feedback to ensure accuracy. For correct CAD programs, we predefine feedback in Table 8.

4 Proposed Method

Given the reference image and the potentially inconsistent CAD program, our goal is to detect and correct program errors. It requires recognizing geometric components and performing spatial geometric operations within the CAD program. To this end, we propose a multimodal large language model (MLLM)-based framework, CAD Program Repairer (ReCAD), which generates helpful feedback for program error detection and then utilizes the feedback for error correction, as illustrated in Fig. 4. Specifically, we build ReCAD by combining the feedback generator ϕ_1 and code editor ϕ_2 with the same MLLM backbone, where ϕ_1 produces descriptions of errors in the CAD program as feedback, offering guidance for ϕ_2 to correct. We train them with two-stage supervised fine-tuning (SFT), followed by reinforcement learning (RL) with reward functions. Each MLLM consists of three key modules: vision encoder, large lan-

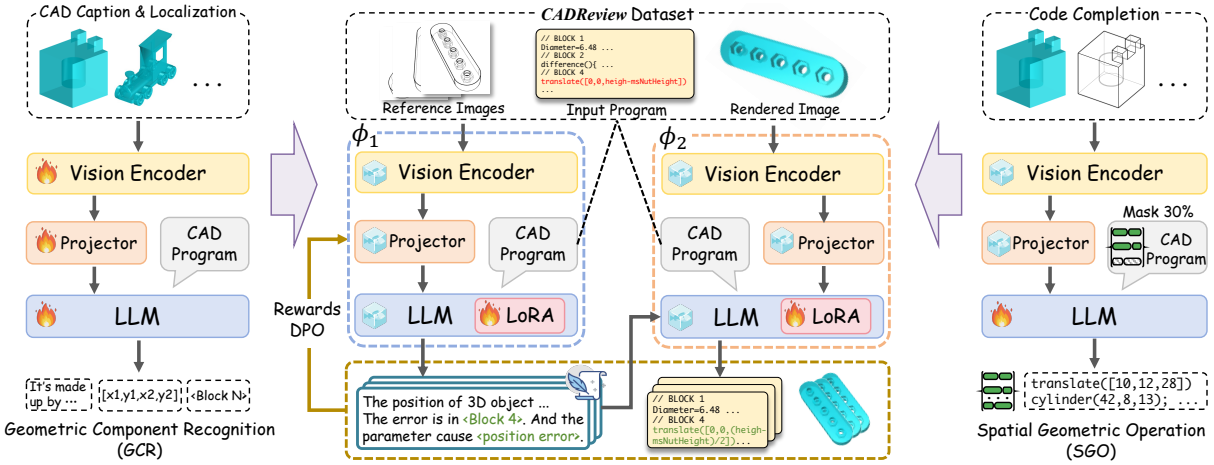


Figure 4: Overview of our ReCAD. We design geometric component recognition (GCR) and spatial geometric operation (SGO) mechanisms to initialize feedback generator ϕ_1 and code editor ϕ_2, respectively.

guage model (LLM) and vision-language projector. Moreover, we also explore different MLLMs (*i.e.*, Qwen2-VL (Wang et al., 2024a) and LLaVA-OV (Li et al., 2024a)) as the backbone for both ϕ_1 and ϕ_2. In the following, we elaborate on the details of our ReCAD framework.

4.1 Feedback Generator

Feedback on the CAD program aims to describe potential discrepancies between the program and reference images. Although existing models can generate feedback for traditional code editing (Chae et al., 2024), they struggle to align geometric components with corresponding code blocks in the CAD domain, leading to ineffective feedback. Therefore, we propose the geometric component recognition (GCR) mechanism to enhance the ability of feedback generator ϕ_1 to recognize both visual and programmatic components.

4.1.1 Geometric Component Recognition

We collect and augment over 700*K* CAD caption and localization data to train ϕ_1 for GCR. More details about the data are provided in Appendix B.

CAD Caption. In contrast to traditional image caption, CAD caption aims to textually describe the components that make up the 3D object, rather than the entire object. Specifically, we use images and beginner-level textual CAD modeling instructions from the Text2CAD (Khan et al., 2024b) dataset as image-text pairs for captioning. The instruction primarily includes shape properties, *e.g.*, a circular CAD model with a central hole, providing guidance for MLLM to visually recognize geometric components. Following (Liu et al., 2024), we only train

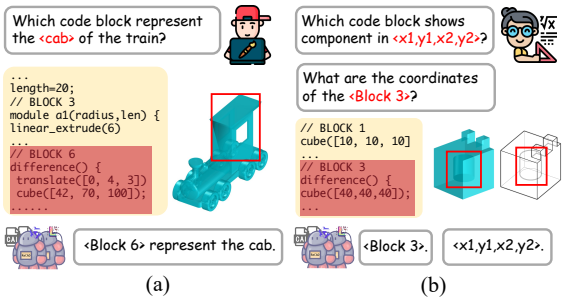


Figure 5: Overview of CAD localization. (a) Semantic matching. (b) Coordinate matching and localization.

the vision-language projector while keeping the vision encoder and large language model (LLM) frozen. After that, the projector effectively refines visual features, enabling the LLM to focus on these geometric components.

CAD Localization. CAD localization aims to align the recognized geometric components with the CAD program. It can be divided into three aspects, as illustrated in Fig. 5. *(i) Semantic matching* predict the code block based on the corresponding specific semantic labels. *(ii) Coordinate matching* queries which code block corresponds to the specific coordinates of the rendered image. *(iii) Coordinate localization* determines the coordinates of a given code block. To build the training data, we first utilize the CADTalk (Yuan et al., 2024) dataset, which provides a semantic label for each code block, *e.g.*, cab of the train in Fig. 5 (a). Since the CAD review task involves evaluating internal components, we define rules and prompt GPT-4o to generate CAD programs comprising basic primitives and boolean operations (*e.g.*, difference and

union), as shown in Fig. 5 (b). In particular, we collect two types of images to assist the model in simultaneously understanding both the reference and rendered images for CAD review. Inspired by the continual pre-training (Chen et al., 2024c), we initialize the ϕ_1 trained on the CAD caption and jointly fine-tune the vision encoder and LLM for CAD localization. After sequential training, we observe that localization performance achieves high accuracy at nearly 90%, while the performance of CAD caption remains unaffected.

4.1.2 SFT for Feedback Generation

After sequential training on CAD caption and localization, we perform supervised fine-tuning (SFT) of ϕ_1 to generate feedback. Specifically, for each input CAD program, we first obtain the rendered image from a specific viewpoint as supplementary input. The reference images, CAD program and rendered image are then fed into frozen ϕ_1 with learnable LoRA (Hu et al., 2022) layers for language modeling. Lastly, we use the cross-entropy loss to maximize the probability of predicting ground truth feedback.

4.2 Code Editor

In addition to recognizing geometric components, the code editor ϕ_2 utilizes the generated feedback as guidance to perform geometric operations for program error correction. We propose the spatial geometric operation (SGO) mechanism for the code editor ϕ_2 to learn spatial relationships and geometric transformations within the CAD programs.

4.2.1 Spatial Geometric Operation

The CAD program mainly consists of operators (*e.g.*, rotation, translation, boolean operations) and operands (*e.g.*, displacement along axes, angles). Due to the lack of explicit scalar values in the input image, the code editor ϕ_2 learns these operations through code completion, where input code provides the necessary reference spatial context. Specifically, we use the same data sources as CAD localization (*i.e.*, CADTalk and LLM-augmented data) and randomly mask 30% code blocks for ϕ_2 to predict. To maintain the geometric component recognition ability, we initialize ϕ_2 by the MLLM trained on the CAD caption and localization.

Through empirical observation, we find that the cross-entropy loss converges rapidly to a minimal value when ϕ_2 is directly trained on code completion. It is because the syntax structures of the code are highly similar, which makes it easier for the model to learn. The loss can not accurately reflect the discrepancies in operand values. Additionally, representing these operand values with decimals in text format is not token-efficient, as they often require multiple tokens for representation. Therefore, we first quantize the spatial position values to 8 bits, resulting in a maximum value of 256 (see Appendix B.4). We apply a re-weighting loss $\mathcal{L}_{sgo}$ by doubling the loss values for numerical tokens to train the LLM of ϕ_2:

$$\mathcal{L}_{sgo} = w_i \cdot \mathcal{L}_i,\ w_i = \begin{cases} 2, & \text{if } y_i \in \mathbb{R}, \\ 1, & \text{otherwise}, \end{cases} \tag{1}$$

where $\mathcal{L}_i$ and w_i denote loss and weight of the i-th token. $\mathbb{R}$ represents the set of real numbers.

4.2.2 SFT for Error Correction

The code editor ϕ_2 takes the reference image, CAD program, rendered image, and generated feedback as inputs. Given that the vision encoder of ϕ_2 is already sufficiently robust in recognizing geometric components, we only employ LoRA fine-tuning on the LLM for program error correction.

4.3 RL-based Refinement

To ensure faithful feedback as guidance for code editing, we design two distinct reward functions to refine the feedback generator ϕ_1. The first reward function leverages error diagnostics to judge the correctness of the generated feedback. The second reward function is defined as the visual similarity between the rendered image of the edited code and the reference image. We further explore a time-consuming reward function based on 3D point clouds in Appendix C.5.

Error Diagnostic. This diagnostic reward evaluates the correctness of identified code blocks and error types. The reward function $\mathcal{V}_d$ for two distinct states can be formulated as:

$$\mathcal{V}_d = \begin{cases} 1, \text{ if correct feedback}, \\ 0, \text{ otherwise}, \end{cases} \tag{2}$$

where $\mathcal{V}_d$ equals 1 if both the code block and error type in the feedback are correctly identified.

Visual Similarity. We draw inspiration from self-similarity (Belouadi et al., 2024) and argue that the model can self-assess the similarity between the input reference image and the rendered output image. Thus, we obtain their visual features by the vision

Method	Machine-made Program							Human-made Program						
	R_L↑	BS↑	Acc↑	CD↓	MMD↓	JSD↓	IR↓	R_L↑	BS↑	Acc↑	CD↓	MMD↓	JSD↓	IR↓
Claude 3.5	24.84	59.96	32.19	4.06	1.66	36.16	32.97	24.88	60.23	21.51	9.03	2.63	79.63	86.09
Gemini 2.0	27.08	63.29	36.83	3.96	0.45	11.49	27.10	25.61	59.95	23.36	5.97	1.60	55.85	34.88
GPT-4o	31.31	66.25	41.54	4.34	0.56	12.07	28.43	29.97	64.81	31.84	5.52	1.55	48.70	18.57
Llama 3.2†	32.48	62.24	56.23	2.71	0.43	5.44	4.01	32.65	67.23	51.24	5.82	1.67	36.18	16.14
ReCAD-LA†	**35.89**	**70.24**	**73.11**	1.45	0.32	3.42	**0.00**	33.14	67.43	59.15	4.76	0.75	33.09	13.74
ReCAD-QW†	35.62	70.01	71.83	**1.43**	**0.30**	**2.80**	**0.00**	**33.20**	**67.47**	**63.60**	**4.42**	**0.52**	**30.87**	**10.59**

Table 1: Main results of baselines and our ReCAD. †: fine-tuning on *CADReview* dataset. CD, MMD and JSD are multiplied by 10^3. **Bold**: best results.

ReCAD	w/o	Machine-made			Human-made		
		Acc↑	CD↓	JSD↓	Acc↑	CD↓	JSD↓
LA	GCR	67.43	2.08	4.86	51.75	5.86	37.66
	SGO	70.55	1.93	4.03	58.79	6.84	39.24
	Fed	-	1.84	4.41	-	5.71	35.40
	Red	71.16	1.57	3.58	58.01	5.27	33.58
	-	**73.11**	**1.45**	**3.42**	**59.15**	**4.76**	**33.09**
QW	GCR	66.28	2.03	4.41	53.74	6.03	36.05
	SGO	70.22	1.95	4.27	60.82	8.08	42.56
	Fed	-	1.59	3.62	-	5.49	32.30
	Red	71.26	1.80	2.95	61.79	4.74	31.96
	-	**71.83**	**1.43**	**2.80**	**63.60**	**4.42**	**30.87**

Table 2: Results of ablation study. Fed: feedback. Red: reward functions. CD and JSD are multiplied by 10^3. **Bold**: best results.

encoder of ϕ_1 and calculate the cosine similarity of two features as the visual reward function $\mathcal{V}_v$.

DPO. We randomly choose 2,000 training samples and employ top-p sampling method to produce K output feedback. Next, we collect them as rank pairs based on the two reward functions. We optimize ϕ_1 with the direct preference optimization (DPO) (Rafailov et al., 2023) algorithm. More details about the reward functions and refinement process are shown in Appendix C.5.

5 Experiment

5.1 Experimental Setting

Implementation Details. All experiments are conducted with four NVIDIA A100-80GB GPUs. We implement the ReCAD using different multimodal large language models (MLLMs), *i.e.*, QWen2-VL (Wang et al., 2024a) and LLaVA-OV (Li et al., 2024a), each with approximately 7 billion parameters. These variants of ReCAD are named as ReCAD-QW and ReCAD-LA, respectively. For the feedback generator ϕ_1, we first only train the projector for 1 epoch on CAD caption. Next, we keep the projector frozen and fully fine-tune the large language model (LLM) and vision encoder for 1 epoch on CAD localization. For feedback generation, we then apply learnable LoRA layers into the LLM of ϕ_1, setting the rank to 8, a learning rate of 1e-5, 3 training epochs and a maximum sequence length of 3072. For code editor ϕ_2, we initialize it using the one trained on CAD caption and localization. We train the LLM of ϕ_2 on spatial geometric operation for 1 epoch. Based on the generated feedback from ϕ_1, we subsequently perform LoRA fine-tuning on ϕ_2 to edit erroneous code, with the rank of 64. The batch size is 4 and we train 3 epochs with a learning rate of 4e-5. Moreover, the maximum sequence length of ϕ_2 is expanded to 8192. During the RL-based refinement, we use a top-p sampling strategy on ϕ_1 with a temperature of 0.8 and p=0.9, generating 8 feedback samples for each input. Finally, we maintain those feedback pairs where the diagnostic reward of chosen feedback equals 1, and the visual reward difference exceeds 0.25 for direct preference optimization.

Evaluation Metrics. Similar to previous studies on code review (Chae et al., 2024; Tang et al., 2024), we employ metrics on feedback generation and program error correction to comprehensively evaluate the performance of our ReCAD and baselines. Following (Chen et al., 2023, 2024a), we adopt natural language generation metrics, *i.e.*, ROUGE_L (R_L) (Lin, 2004), BERTScore (BS) (Zhang et al., 2020), and accuracy (Acc) to assess the quality of generated feedback. Notably, accuracy is a rigorous criterion, requiring both the code block and error type to be correctly predicted. Following (Haque et al., 2022), we utilize chamfer distance (CD), minimum matching distance (MMD), Jensen-Shannon divergence (JSD), and invalidity ratio (IR) to evaluate the consistency between 3D point clouds of the edited and ground truth 3D objects. We detail these metrics in Appendix D.2.

Baselines. To verify the superiority of our ReCAD, we compare with several advanced closed-source multimodal large language models (MLLMs), *i.e.*, GPT-4o (OpenAI, 2023), Claude 3.5 (Anthropic, 2024), and Gemini 2.0 (Anil et al., 2023). Moreover, we fine-tune a larger open-source MLLM, *i.e.*, Llama-3.2-11B (Dubey et al., 2024) for additional comparison. We also report the results of baselines that directly edit code without feedback and apply the few-shot setting in Appendix C.

5.2 Performance Comparison

Table 1 reports the quantitative evaluation results of baselines and our ReCAD framework on *CADReview* dataset. We have the following observations: **(i)** While the leading closed-source MLLM, GPT-4o, remains competitive on natural language generation metrics (*i.e.*, ROUGE$_L$ and BERTScore), it only achieves 41.54% and 31.84% accuracy in generating feedback on human-made and machine-made programs, respectively, lagging behind our ReCAD by over 30%. It suggests that they exhibit similar capabilities in describing visual anomalies, but close-source MLLMs struggle to align CAD programs with visual information for feedback generation, leading to poor performance in identifying errors within CAD programs. Moreover, they often generate non-compilable CAD programs, with an invalid rate exceeding 25%. **(ii)** Our ReCAD-LA and ReCAD-QW consistently outperform the larger open-source Llama 3.2 on all metrics, even though these three models have been fine-tuned on the *CADReview* dataset. For example, on human-made programs, our ReCAD-QW surpasses Llama 3.2 by "+12.36" and "-5.31" on Acc and JSD, respectively. This gap highlights that fine-tuning alone is insufficient for the CAD review task. Our geometric component recognition (GCR) and spatial geometric operation (SGO) mechanisms assist MLLMs in generating accurate feedback and translating geometric operations into precise code implementations. **(iii)** Compared to the machine-made program, all methods experience a performance drop on the human-made program. Our ReCAD still maintains state-of-the-art performance on the human-made program. It demonstrates that ReCAD not only learns geometric operations for simple primitives in machine-made programs but also handles complex geometric components and program structures in human-made programs, resulting in more accurate reconstructed 3D objects. **(iv)** Despite ReCAD-LA achieving higher accuracy

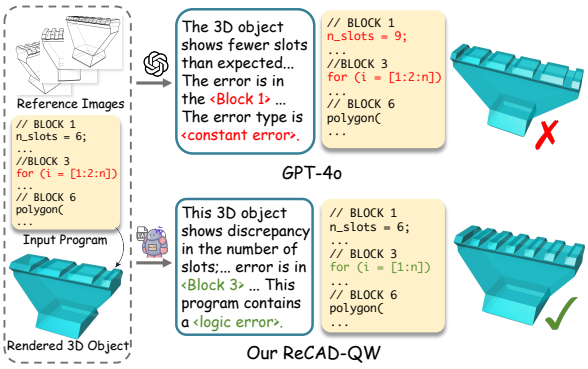


Figure 6: Cases of different methods. The green and red denote correct and incorrect prediction, respectively.

in feedback generation on machine-made programs, ReCAD-QW consistently leads in the quality of reconstructed 3D objects on two types of programs. It shows that the feedback generated by feedback generator ϕ_1 of ReCAD-QW provides more effective guidance for code editor ϕ_2 to correct errors.

5.3 Ablation Study

Table 2 shows ablation experimental results. We find that: **(i)** Removing the geometric component recognition (GCR) mechanism degrades ReCAD's performance on all metrics. The feedback generator ϕ_1 fails to align code blocks with their corresponding components for feedback generation. This misalignment also adversely affects the initialization of code editor ϕ_2, preventing accurate program error correction. As a result, the quality of the reconstructed 3D objects is even inferior to the variant where no feedback is provided to ϕ_2 (*i.e.*, ReCAD w/o Fed). **(ii)** The spatial geometric operation (SGO) mechanism is crucial in handling human-made programs, which often feature more complex geometric components (*e.g.*, hidden internal components) and program structures. It validates that ϕ_2 effectively captures these factors through the SGO mechanism to further edit the erroneous code. **(iii)** By comparing ReCAD w/o Fed, ReCAD w/o Red and ReCAD, we observe that accurate feedback significantly enhances the ability of ϕ_2 to edit code. Furthermore, we notice a slight performance improvement by applying the reward functions $\mathcal{V}_d$ and $\mathcal{V}_v$. They evaluate the quality of both the feedback and the edited code to refine ϕ_1, which ensure cycle consistency between ϕ_1 and ϕ_2.

5.4 Case Study

Fig. 6 shows the generated feedback, edited code and corresponding reconstructed 3D object by GPT-

4o and our ReCAD-QW. Intuitively, our ReCAD-QW produces a more grounded 3D object. Specifically, although both GPT-4o and ReCAD-QW correctly identify the visual discrepancies regarding the number of slots in the 3D object, our ReCAD-QW successfully locates the erroneous loop statement in block 6 for program error correction. In contrast, GPT-4o relies solely on the semantics of the variable name, which results in the edited code containing even more errors.

6 Conclusion

In this paper, we introduce the CAD review task to automatically detect and correct errors in CAD programs based on reference images. To tackle this task, we propose the *CADReview* dataset and ReCAD framework to detect program errors and generate helpful feedback on error correction. Extensive experiments show that ReCAD outperforms existing MLLMs, providing an effective solution for the AI-aided review process in industrial design.

7 Limitations

In this paper, we introduce the CAD review task. We also create the *CADReview* dataset and propose the ReCAD framework for this task. Although our framework aids designers in the design review process, it does not consider the optimal code editing solution. Given the diversity in programmatic realizations of the same geometric components, the optimal solution is hard to define. Additionally, due to limitations about copyright and data availability of CAD programming languages, our *CADReview* dataset currently only includes open-source OpenSCAD code. Since all CAD programming languages adhere to common standards (*e.g.*, basic geometric primitives and operations), we believe the ReCAD framework is language-agnostic and can be adapted to any CAD programming language in the future.

Acknowledgments

This research is supported by Guangdong Provincial Natural Science Foundation for Outstanding Youth Team Project (2024B1515040010), the Fundamental Research Funds for the Central Universities, South China University of Technology (x2rjD2240100), Guangdong Provincial Fund for Basic and Applied Basic Research—Regional Joint Fund Project (Key Project) (2023B1515120078).

References

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Jonas Belouadi, Simone Paolo Ponzetto, and Steffen Eger. 2024. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. *CoRR*, abs/2405.15306.

Hyungjoo Chae, Taeyoon Kwon, Seungjun Moon, Yongho Song, Dongjin Kang, Kai Tzu-iunn Ong, Beong-woo Kwak, Seonghyeon Bae, Seung-won Hwang, and Jinyoung Yeo. 2024. Coffee-gym: An environment for evaluating and improving natural language feedback on erroneous code. In *Proc. of EMNLP*, pages 22503–22524. Association for Computational Linguistics.

Kuang-Hua Chang. 2015. *e-Design: computer-aided engineering design*. Academic Press.

Jiali Chen, Zhenjun Guo, Jiayuan Xie, Yi Cai, and Qing Li. 2023. Deconfounded visual question generation with causal inference. In *Proc. of ACM MM*, pages 5132–5142. ACM.

Jiali Chen, Xusen Hei, Yuqi Xue, Yuancheng Wei, Jiayuan Xie, Yi Cai, and Qing Li. 2024a. Learning to correction: Explainable feedback generation for visual commonsense reasoning distractor. In *Proc. of ACM MM*, pages 8209–8218. ACM.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024b. Teaching large language models to self-debug. In *Proc. of ICLR*. OpenReview.net.

Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, Lixin Gu, Xuehui Wang, Qingyun Li, Yimin Ren, Zixuan Chen, Jiapeng Luo, Jiahao Wang, Tan Jiang, Bo Wang, Conghui He, Botian Shi, Xingcheng Zhang, Han Lv, Yi Wang, Wenqi Shao, Pei Chu, Zhongying Tu, Tong He, Zhiyong Wu, Huipeng Deng, Jiaye Ge, Kai Chen, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin,

Yu Qiao, Jifeng Dai, and Wenhai Wang. 2024c. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *CoRR*, abs/2412.05271.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Proc. of EMNLP Findings*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: large language models can self-correct with tool-interactive critiquing. In *Proc. of ICLR*. OpenReview.net.

Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Sheikh Iqbal Ahamed. 2022. Deepcad: A standalone deep neural network-based framework for classification and anomaly detection in smart healthcare systems. In *Proc. of ICCV*, pages 218–227. IEEE.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *Proc. of ICLR*. OpenReview.net.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *CoRR*, abs/2401.04088.

R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2023. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Trans. Graph.*, 42(4):49:1–49:17.

Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. 2024a. Cad-signet: CAD language inference from point clouds using layer-wise sketch instance guided attention. In *Proc. of CVPR*, pages 4713–4722. IEEE.

Mohammad Sadil Khan, Sankalp Sinha, Sheikh Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. 2024b. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. In *Proc. of NeurIPS*.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. 2024a. Llava-onevision: Easy visual task transfer. *CoRR*, abs/2408.03326.

Feng Li, Renrui Zhang, Hao Zhang, Yuanhan Zhang, Bo Li, Wei Li, Zejun Ma, and Chunyuan Li. 2024b. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *CoRR*, abs/2407.07895.

Pengxiang Li, Zhi Gao, Bofei Zhang, Tao Yuan, Yuwei Wu, Mehrtash Harandi, Yunde Jia, Song-Chun Zhu, and Qing Li. 2024c. Fire: A dataset for feedback integration and refinement evaluation of multimodal models. In *Proc. of NeurIPS*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Proc. of ACL Workshop*, page 74–81.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved baselines with visual instruction tuning. In *Proc. of CVPR*, pages 26286–26296. IEEE.

Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. 2019. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proc. of CVPR*, pages 909–918. Computer Vision Foundation / IEEE.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proc. of ACL*, pages 311–318.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Proc. of NeurIPS*.

Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. 2019. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proc. of CVPR*, pages 658–666. Computer Vision Foundation / IEEE.

Habib Slim and Mohamed Elhoseiny. 2024. Shapewalk: Compositional shape editing through language-guided chains. In *Proc. of CVPR*, pages 22574–22583. IEEE.

Xunzhu Tang, Kisub Kim, Yewei Song, Cedric Lothritz, Bei Li, Saad Ezzini, Haoye Tian, Jacques Klein, and Tegawendé F. Bissyandé. 2024. Codeagent: Autonomous communicative agents for code review. In *Proc. of EMNLP*, pages 11279–11313. Association for Computational Linguistics.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024a. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *CoRR*, abs/2409.12191.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. 2024b. CAD-GPT: synthesising CAD construction sequence with spatial reasoning-enhanced multimodal llms. *CoRR*, abs/2412.19663.

Wikipedia. 2025a. OpenSCAD — Wikipedia, the free encyclopedia.

Wikipedia. 2025b. Rhinoceros 3D — Wikipedia, the free encyclopedia.

Sifan Wu, Amir Hosein Khasahmadi, Mor Katz, Pradeep Kumar Jayaraman, Yewen Pu, Karl D. D. Willis, and Bang Liu. 2024. Cadvlm: Bridging language and vision in the generation of parametric CAD sketches. In *Proc. of ECCV*, volume 15128 of *Lecture Notes in Computer Science*, pages 368–384. Springer.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. 2022. Skexgen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *Proc. of ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 24698–24724. PMLR.

Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy J. Mitra, and Changjian Li. 2024. Cadtalk: An algorithm and benchmark for semantic commenting of CAD programs. In *Proc. of CVPR*, pages 3753–3762.

Li Yuan, Yi Cai, Jin Wang, and Qing Li. 2023. Joint multimodal entity-relation extraction based on edge-enhanced graph alignment network and word-pair relation tagging. In *Proc. of AAAI*, volume 37, pages 11051–11059.

Li Yuan, Yi Cai, Jingyu Xu, Qing Li, and Tao Wang. 2025. A fine-grained network for joint multimodal entity-relation extraction. *IEEE Transactions on Knowledge and Data Engineering*, 37(1):1–14.

Xiang Yue, Yuansheng Ni, Tianyu Zheng, Kai Zhang, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. In *Proc. of CVPR*, pages 9556–9567. IEEE.

Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. 2023. Self-edit: Fault-aware code editor for code generation. In *Proc. of ACL*, pages 769–787. Association for Computational Linguistics.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with BERT. In *Proc. of ICLR*. OpenReview.net.

A *CADReview* Dataset Construction

In this section, we provide additional details about the construction of our *CADReview* dataset. Fig. 7 shows the demonstration of diverse 3D objects in our *CADReview* dataset.

A.1 CAD Program Collection

The source data of the human-made programs are collected from online design communities[2]. We manually remove comments and redundant code segments to simplify the program structure. Our dataset consists of 21,949 samples, including 9,218 human-made programs and 12,731 machine-generated programs. It is worth noting that each set (*i.e.*, training or testing set) contains CAD programs from a set of mutually exclusive 3D objects. Therefore, the evaluation is conducted on previously unseen 3D objects, which allows for a better assessment of the model's generalization ability. Fig. 10 illustrates samples of human-made and machine-made programs in our dataset. The distribution of error types is presented in Fig. 8.

A.2 Reference Image Collection

We utilize 3D modeling software (*i.e.*, Rhino) and adopt pen-style mode for rendering to obtain the reference images. Specifically, for these 3D objects, we also employ a perspective view with the pen-style mode for rendering, allowing the potential internal components to be visible. Each viewpoint of the reference image is selected from the range of $[0, 2\pi)$, with the condition that the angular difference between consecutive viewpoints must be greater than $\pi/10$. Once the reference images are selected, they are provided to another annotator to evaluate whether these images sufficiently capture the essential structure of the 3D object.

A.3 Error Creation

We develop a graphical annotation interface for annotators to create errors on the collected CAD programs, as shown in Fig. 9. Our annotation team consists of 10 experienced designers familiar with OpenSCAD code for error annotation.

A.4 Feedback Annotation

We provide the prompt for GPT-4o to annotate feedback in Table 7. Furthermore, for CAD programs that are consistent with the reference images (*i.e.*, correct programs), we randomly select one of the 10 predefined feedback options listed in Table 8.

B Additional Training Dataset

To construct training data for CAD caption and CAD localization, we utilize the existing Text2CAD (Khan et al., 2024b) and CADTalk (Yuan et al., 2024) datasets. Considering that 3D objects from realistic industrial design often feature hidden internal components, we further augment data by GPT-4o.

B.1 Text2CAD

Text2CAD dataset is built on the original DeepCAD (Haque et al., 2022) dataset. It leverages large language models (LLMs) and vision-language models (VLMs) to generate text prompts, enabling the translation from textual descriptions to parametric CAD models. Initially, LLaVA-NeXT (Li et al., 2024b) is employed to produce shape descriptions of these CAD models and their intermediate components. Subsequently, Mixtral-50B (Jiang et al., 2024) generates multi-level textual modeling instructions based on the shape descriptions and design details provided in DeepCAD. We utilize beginner-level instructions that describe how CAD models are composed of simple components. Meanwhile, for each instruction, we select four different views of the corresponding image as training data, effectively multiplying the total number of training samples by four. It results in over $600K$ training samples for CAD caption. Moreover, we prompt the MLLM to describe the composition of the 3D object during training for CAD caption.

B.2 CADTalk

CADTalk (Yuan et al., 2024) is a benchmark dataset for semantic comment annotation of CAD programs, where each code block is annotated to represent the semantic components of 3D objects. For example, Block 4 represents the hand of the 3D object. It contains approximately $6.5K$ samples, most of which are derived from labeled 3D objects in the PartNet (Mo et al., 2019) dataset and converted into OpenSCAD programs. Another portion is collected from online repositories (*e.g.*., Thingiverse[3]. We leverage CADTalk to ground the coordinates of each semantic label in rendered images for semantic matching in CAD localization. For coordinate matching and localization, we randomly com-

[2]https://cults3d.com

[3]https://www.thingiverse.com

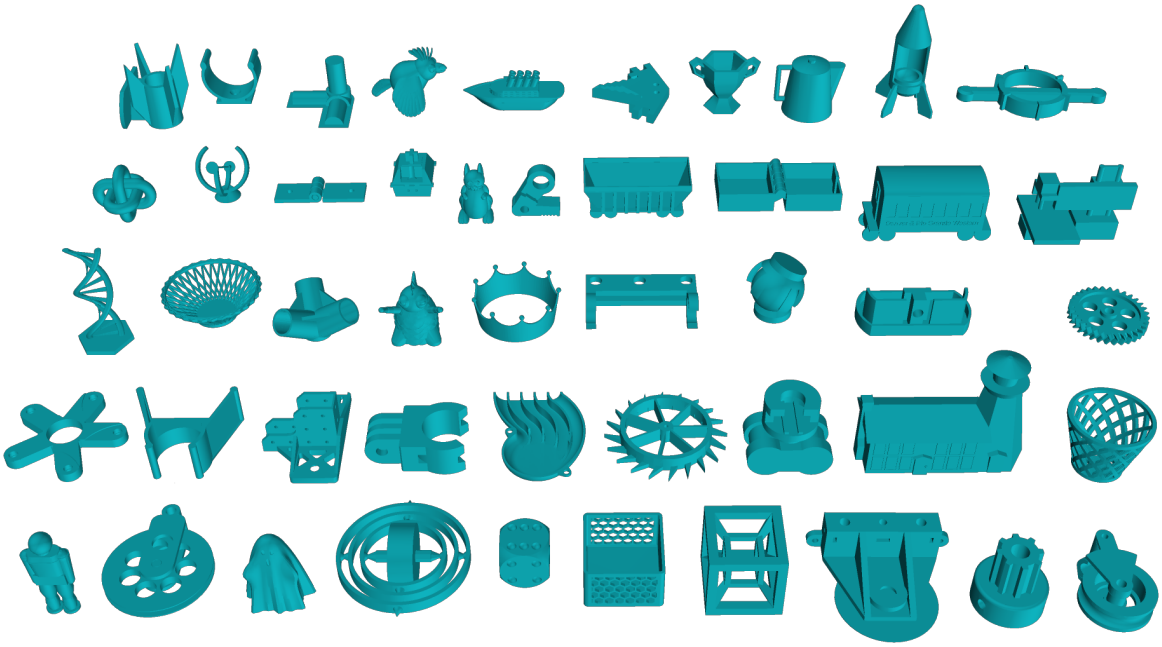


Figure 7: Demonstration of various 3D objects from our *CADReview* dataset.

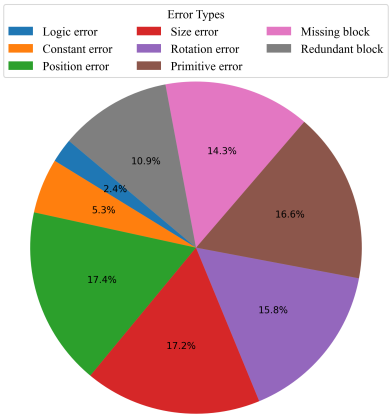


Figure 8: Distribution statistics of error types in our *CADReview* dataset.

bine code blocks from the same sample program in CADTalk, resulting in over 10K data samples.

B.3 LLM-Augmented Data

Considering that real 3D objects often feature complex internal components, we further synthesize LLM-augmented data using boolean operations (*e.g.*, difference and union) with GPT-4. Subsequently, we design rules to randomly modify the values of the generated data for data augmentation. Finally, we obtain over 10K data samples. In particular, we collect two types of rendered images using both the pen-style and physical rendering modes with a perspective view, which enhances the model's generalization ability. This LLM-augmented data is used for coordinate matching and localization in CAD localization. As shown in Fig. 5 (b), these augmented programs consist of basic primitives and boolean operations without representation of tangible objects. The prompt for GPT-4o is provided in Table 9.

B.4 Value Quantization

Considering that CADTalk and LLM-augmented data often contain many decimal values, MLLMs predict continuous parameters through regression, and slight inaccuracies can easily violate these critical constraints. As a result, it is challenging for basic MLLMs to understand spatial relationships within CAD programs for spatial geometric operations (SGO). Following previous studies (Khan et al., 2024a,b), we quantize the spatial position values (*i.e.*, axis coordinates and translation offset values) of the CAD programs to 8 bits, yielding a maximum value of 256.

B.5 Performance Evaluation

After introducing the training data, we briefly present the performance of our ReCAD on these tasks, which demonstrates the effectiveness of our geometric component recognition (GCR) and spatial geometric operations (SGO) mechanisms.

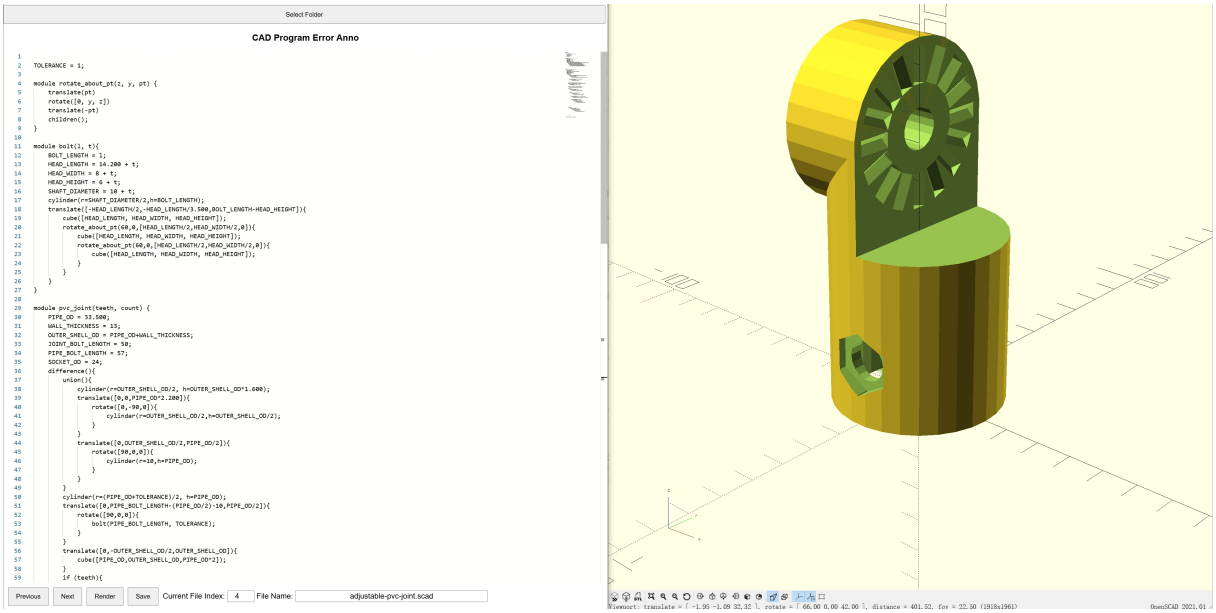
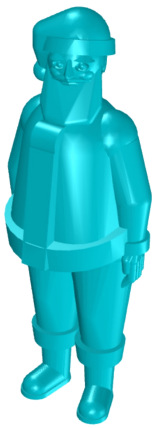
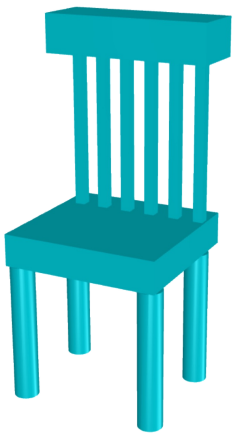


Figure 9: Graphic annotation interface for error creation.

```
module m1(){
module m2(){
hull(){
sphere(2);
...
m2();
...
module m3(){
hull(){
cylinder(2,6.200,6.100);
...
difference(){
...
module m4(){
hull(){
translate([-10,12,28])
cylinder(42,8,13);
...
module m5(){
...
difference(){
translate([0,5,115])
resize(newsize=[20,22,25])
sphere(10);
...
```



```
translate([-2, 0, 13])
rotate([0, 0, 0])
cube([77, 73, 14],
center=true);
translate([-35, 20, -50])
rotate([0, 0, 0])
cylinder(h=60, r1=6, r2=6,
center=true, $fn=30);
...
translate([-6, 21, 28])
rotate([0, 90, 0])
cube([70, 5, 5],
center=true);
translate([7, 21, 28])
rotate([90, 0, 0])
cube([5, 70, 5],
center=true);
translate([20, 21, 28])
rotate([0, 0, 90])
cube([5, 5, 70],
center=true);
translate([33, 21, 28])
rotate([0, 0, 10])
cube([5, 5, 70],
center=true);
...
```



(a) (b)

Figure 10: Samples from *CADReview* dataset. (a) Human-made program. (b) Machine-made program.

Specifically, for CAD caption, CAD localization, and code completion, we sample 500 instances from each task for performance evaluation.

For CAD caption, we observe a significant improvement in performance on natural language generation metrics (*e.g.*, BLEU-4 (Papineni et al., 2002) and BERTScore (Zhang et al., 2020)). For instance, after training the model on CAD caption, the BLEU-4 score increases from 4.42 to 20.03. It suggests that the projector effectively refines visual features, enabling our ReCAD to focus on geometric components. For CAD localization, we set the threshold of Intersection over Union (IoU) (Rezatofighi et al., 2019) between the predicted and ground-truth bounding boxes to 0.8, considering predictions with an IoU below this value as failures. The overall accuracy of our ReCAD-LA and ReCAD-QW are 91.06% and 88.25%, respectively. After sequential training on CAD caption and localization, we notice that the performance of CAD caption remains unaffected. It demonstrates that our ReCAD framework effectively aligns identified geometric components with their corresponding code blocks. Finally, for code completion in spatial geometric operations (SGO), the trained model achieves a chamfer distance of less than 1 between the 3D object generated from the completed code and the ground truth, whereas the initial model is unable to generate compilable code. It suggests that the SGO mechanism enables our ReCAD framework to effectively learn spatial operations of basic primitives and facilitates further code editing in the CAD review task.

C Additional Results

C.1 Human Evaluation

Considering that the same geometric shape can be realized through different programmatic implementations, we evaluate the code editing solutions of our ReCAD and the baselines by comparing them to those of human designers with human evaluation. We first randomly select 200 samples with a chamfer distance smaller than 5 in human-made programs from our ReCAD and Llama 3.2 prediction results, and both predicted code blocks and error types are correct. The selected samples are consistent across all methods. We invite five volunteer designers familiar with OpenSCAD to evaluate based on the following criteria: Readability (**Red**) measures how clear and understandable is the edited code to a human designer. Reusability (**Reu**) refers to what extent can the edited code be reused or extended to other 3D objects. Preservation **(Pre)** represents how well the edited code maintains the original functionality while eliminating errors. The scores on a scale from 0 to 2, with higher values indicating greater alignment with human-edited code. Table 3 shows the results of our ReCAD and Llama 3.2. The standard deviations of each human evaluation metric confirms the faithfulness of our results. We find that: **(i)** After fine-tuning, these methods demonstrate consistent performance in generating readable code, suggesting that they have effectively learned the syntax rules of CAD programs. **(ii)** Our ReCAD significantly outperforms the Llama 3.2 model in terms of code reusability and functionality preservation, demonstrating that our mechanisms for geometric component recognition (GCR) and spatial geometric operations (SGO) enable more adaptable CAD program generation.

Method	Red	Reu	Pre
Llama 3.2	1.36/0.21	0.94/0.27	0.87/0.26
ReCAD-LA	**1.38**/0.30	1.25/0.21	1.23/0.22
ReCAD-LA	1.31/0.19	**1.29**/0.18	**1.28**/0.24

Table 3: Human evaluation results. Each value is presented as τ/ρ, where τ is the metric value and ρ is the standard deviation. **Bold**: the maximum value.

C.2 Few-Shot Setting for Baselines

In Table 4, we report the few-shot results for close-source MLLM baselines (*i.e.*, Claude 3.5, Gemini 2.0 and GPT-4o). Specifically, we randomly select 200 samples in machine-made and human-made programs, respectively. We utilize CodeBERT (Feng et al., 2020) to encode CAD programs and compute the cosine similarity between the query program and each sample in the training set to find the most similar one as the 1-shot in-context example. We find that the fluctuations in the results are minimal, suggesting that the few-shot approach does not effectively address the challenges of the CAD review task.

C.3 Direct Code Editing for Baselines

Table 5 shows the results for close-source MLLMs to directly editing CAD code without feedback generation. Interestingly, the absence of feedback leads to performance improvements for both GPT-4o and Gemini 2.0. This contrasts with the findings from our ReCAD ablation study in Table 2, where

Model	Few-shot	Machine-made		Human-made	
		Acc↑	CD↓	Acc↑	CD↓
Claude 3.5	✗	35.00	**3.94**	**24.00**	**9.76**
	✓	**36.00**	3.90	22.50	9.73
Gemini 2.0	✗	38.50	3.57	**26.00**	**6.01**
	✓	**40.50**	**3.81**	**26.00**	5.99
GPT-4o	✗	43.50	**4.16**	34.00	5.58
	✓	**45.00**	4.09	**35.00**	**5.70**

Table 4: Few-shot results for close-source MLLMs. CD is multiplied by 10^3. **Bold**: the maximum value.

Model	Feedback	Machine-made		Human-made	
		CD ↓	JSD↓	CD↓	JSD↓
Claude 3.5	✗	4.46	36.75	9.05	82.18
	✓	**4.06**	**36.16**	**9.03**	**79.63**
Gemini 2.0	✗	**3.88**	**9.54**	**4.89**	**53.75**
	✓	3.96	11.49	5.97	55.85
GPT-4o	✗	**3.97**	**8.94**	**5.23**	**45.94**
	✓	4.34	12.07	5.52	48.70

Table 5: Direct code editing results for close-source MLLMs. CD and JSD are multiplied by 10^3. **Bold**: the maximum value.

the lack of feedback results in a significant performance decline for ReCAD. This suggests that closed-source MLLMs are unable to effectively utilize feedback to enhance their ability to edit CAD programs, which is in contrast to code review tasks (Chae et al., 2024) on commonly used programming languages such as C++ and Python.

C.4 Influence of Rendered Image

In our main experiment, we compile the input CAD programs and render an image as supplementary input information. This rendered image is easily obtained and feasible from an implementation perspective. Moreover, our preliminary experiments show that omitting the rendered image results in nearly a 5% drop in feedback accuracy for human-made programs. By analyzing the cases of prediction failure, we find that, without the rendered image, the model struggles to correctly identify erroneous code blocks in 3D objects with more than 10 geometric components. It indicates that the rendered image also plays an important role in aligning components, which is consistent with our motivation of geometric component recognition. Further experiments with multiple viewpoints of the rendered show minimal performance improvement for ReCAD, suggesting that a single rendered image is sufficient as supplementary input.

ReCAD	Reward	Machine-made			Human-made		
		Acc↑	CD↓	JSD↓	Acc↑	CD↓	JSD↓
LA	-	71.16	1.57	3.58	58.01	5.27	33.58
	$\mathcal{V}_d$	72.49	1.52	3.49	59.00	4.85	33.43
	$\mathcal{V}_v$	71.89	1.58	3.56	58.39	4.91	33.54
	$\mathcal{V}_p$	**72.75**	**1.46**	**3.47**	**59.41**	**4.68**	**32.59**
QW	-	71.26	1.80	2.95	61.79	4.74	31.96
	$\mathcal{V}_d$	71.52	1.45	2.84	62.88	4.46	31.11
	$\mathcal{V}_v$	71.49	1.51	2.91	62.20	4.60	31.87
	$\mathcal{V}_p$	**72.09**	**1.38**	**2.78**	**63.16**	**4.44**	**31.18**

Table 6: Results of different reward functions. CD and JSD are multiplied by 10^3. **Bold**: best results.

C.5 Reward Functions

In our RL-based refinement process, we design three reward functions to refine the feedback generator ϕ_1. The first reward function $\mathcal{V}_d$ leverages error diagnostics to judge the correctness of the generated feedback. The second reward function $\mathcal{V}_v$ is defined as the visual similarity between the rendered image of the edited code and the reference image. The third reward function $\mathcal{V}_p$ is based on the sampling 3D point clouds. During the RL-based refinement, we use a top-p sampling strategy on ϕ_1 with a temperature of 0.8 and p=0.9, generating 8 feedback samples for each input. Table 6 shows the comparison of different reward functions. For $\mathcal{V}_d$ and $\mathcal{V}_v$, we maintain those feedback pairs for DPO, where the diagnostic reward of chosen feedback equals 1, or the visual reward difference exceeds 0.25. For $\mathcal{V}_p$, we retain feedback pairs where the generated candidate 3D objects have the smallest chamfer distance to the ground truth, as the chosen samples. Conversely, the candidates with the largest chamfer distance to the ground truth are treated as reject samples. We observe that, although $\mathcal{V}_p$ yields the best performance improvement for our ReCAD, the process of sampling 3D point clouds and calculating distances is time-consuming. Therefore, we only present the results with $\mathcal{V}_d$ and $\mathcal{V}_v$ in the main experiment.

Moreover, consistent with the finding in code review for generic programming languages (Chae et al., 2024), we find that high-quality feedback is crucial for successful code editing. Therefore, we apply DPO solely to ϕ_1. Our attempts to apply it to ϕ_2 show no performance improvement. We believe that further improvements in CAD code editing will require larger and more diverse pre-trained CAD code generation datasets in the future.

D More Experimental Details

D.1 Baseline Implementation

The prompt for close-source MLLMs (*i.e.*, GPT-4o, Gemini 2.0, Claude 3.5) is presented in the table 10. Moreover, for a fair comparison, we also apply LoRA fine-tuning on the open-source Llama-3.2-11B model, with a LoRA rank of 8, a learning rate of 1e-5, 3 training epochs, a maximum sequence length of 8192, and a batch size of 2. For code editing, LoRA fine-tuning is conducted on another Llama 3.2 model with a LoRA rank of 64, 3 training epochs, a maximum sequence length of 8192 and a batch size of 1.

D.2 Evaluation Metrics

Following (Haque et al., 2022), we adopt four common evaluation metrics to assess the quality of reconstructed 3D objects by edited CAD programs. While these metrics are originally designed for evaluating point cloud generation, we first convert these 3D objects into point clouds. These metrics are then computed by comparing a set of reference shapes $\mathcal{S}$ with a set of generated shapes $\mathcal{G}$. To better evaluate the outliers of reconstructed 3D objects, we report the mean chamfer distance (CD) scores in our experiments.

Chamfer distance (CD) is a metric used to evaluate the similarity between two point clouds, P and Q. Specifically, P represents the reference point cloud (*i.e.*, ground truth shape) and Q represents the generated point cloud (*i.e.*, reconstructed shape). It calculates the average squared distance from each point $p \in P$ to its nearest point $q \in Q$, and vice versa. The formula computes the minimum squared Euclidean distance $\|p-q\|^2$ for each point $p \in P$ to its closest neighbor in Q, and similarly for each point $q \in Q$ to its closest neighbor in P. CD can be defined as the sum of these minimum distances and normalized by the number of points in each set, providing a measure of how well the generated point cloud approximates the reference point cloud:

$$\begin{aligned}\mathrm{CD}(P,Q) =& \frac{1}{|P|}\sum_{p\in P}\min_{q\in Q}\|p-q\|^2 \\ &+ \frac{1}{|Q|}\sum_{q\in Q}\min_{p\in P}\|q-p\|^2.\end{aligned} \tag{3}$$

Minimum matching distance (MMD) quantifies the fidelity of the generated shapes. It is calculated by determining the Chamfer Distance between each shape in the reference set $\mathcal{S}$ and its closest counterpart in the generated set $\mathcal{G}$. MMD can be defined as the average over all the nearest distances:

$$\mathrm{MMD}(\mathcal{S},\mathcal{G}) = \frac{1}{|\mathcal{S}|}\sum_{Y\in\mathcal{S}}\min_{X\in\mathcal{G}} d^{CD}(X,Y). \tag{4}$$

Jensen-Shannon divergence (JSD) is a statistical measure used to quantify the difference between two probability distributions. It is employed to assess the similarity between the reference set S and the generated set G by computing the marginal point distributions. The JSD can be mathematically expressed as:

$$\begin{aligned}\mathrm{JSD}(P_S,P_G) =& \frac{1}{2}D_{\mathrm{KL}}(P_S \parallel M) \\ &+ \frac{1}{2}D_{\mathrm{KL}}(P_G \parallel M),\end{aligned} \tag{5}$$

where $M = \frac{1}{2}(P_S + P_G)$. D_{KL} denotes the Kullback-Leibler (KL) divergence. The marginal distributions P_S and P_G are approximated by discretizing the space into 1024 voxel grids and assigning each point in the point clouds to one of these grids.

Invalid ratio (IR) is calculated to measure the proportion of invalid CAD programs, which fail to compile for 3D object generation.

Prompt: Please generate feedback for the 3D model based on the following information to support the code review process.
- The first picture is the correct design drawing, and the second picture is the 3D model rendered by OpenSCAD code.
- Error message: There is a Position anomaly on the code.
- The length of comment: No more than 75 words.
Requirements of Feedback Generation:
1. First briefly describe the visual anomaly: clearly point out which component of the 3D model on the second picture is visually inconsistent with the design drawing. The description should focus on the deviation of the specific components in the model.
2. Then describe the error in the code: explain the specific problem in the code that caused the visual anomaly. Do not give the correct code directly, but explain what is incorrect and point out the correct situation.
3. Finally, point out the location of the erroneous code segment in the erroneous code to facilitate subsequent modifications.
The erroneous code segment is as follows: `<erroneous_code_seg>`
The correct code should be: `<correct_code_seg>`
The full erroneous code is as follows: `<full_erroneous_code>`

Table 7: The prompt for GPT-4o to annotate feedback on CAD programs.

1. The 3D rendering captures the essence of the design blueprint with remarkable precision and fidelity.
2. The 3D model matches the design drawing perfectly, with no deviations in key features like frames and recesses.
3. The OpenSCAD-generated 3D model matches the original design drawing perfectly in all aspects.
4. The 3D model mirrors the design drawing with exceptional clarity, maintaining all specified features.
5. The implementation of the design in OpenSCAD results in a highly accurate and detailed 3D model.
6. The alignment between the 3D rendering and the design drawing is precise, with all features correctly placed.
7. The design intent is fully realized in the 3D model, with precise implementation of all structural elements.
8. The faithful replication of the design drawing in the 3D model indicates precise coding and attention to detail.
9. The correspondence between the design plan and the 3D model is seamless, with no misalignment or deviation.
10. A careful analysis shows the 3D model to be a perfect reproduction of the design drawing.

Table 8: The list of predefined feedback for correct CAD programs.

Prompt: Please generate a piece of OpenSCAD code that uses the boolean operations, such as `difference()`, `union()` and `intersection()` to construct a 3D model. The model should combine multiple geometric shapes, such as spheres, cubes, and cylinders to showcase the effect of the convex hull operation. Ensure that the internal component has a certain level of hierarchy and interlacing, and enhance the internal details. The code should be written in a modular manner for easy modification and adjustment, with clear readability and appropriate comments.

Table 9: The prompt for LLM-augmented data annotation for CAD localization.

Prompt: You are an expert in CAD design. You will be given correct reference images from multiple viewpoints and CAD program with the potential discrepancy, and your task is to identify the error type based on the given images and CAD program, write down the error, feedback and generate the correct code.
The error type must be one of the following:
No error: The code shown does not have any error
Missing block: A block is missing from the code
Redundant block: a block in the code is redundant
Size error: There is an error in the size of a block, e.g. the radius of a ball, the length of the sides of a cube.
Position error: A block has an incorrect translation or a wrongly added or missing translation.
Rotation error: The rotation angle of a block is incorrect or a rotation is incorrectly added or missing.
Primitive error: the shape of a block is incorrect, e.g. cube becomes cylinder or sphere, cylinder becomes cube or sphere, sphere becomes cube or cylinder Logic error: an if or for condition error in the code.
Constant error: The value of a global variable in the code is wrong.
Requirements of feedback generation:
1. First briefly describe the visual anomaly: point out which component of the 3D model in the second image is visually inconsistent with the first image. The description should focus on the deviation of the specific components in the model.
2. Then describe the error in the code: explain the specific problem in the code that caused the visual anomaly and explain what is incorrect and point out the correct situation.
3. The feedback should be concise and clear, and no more than 75 words.
Example of feedback: The 3D model shows a deviation in the rotation of the component compared to the design drawing. Specifically, the rotation applied in Block 2 has an incorrect third parameter of 330 degrees instead of 0 degrees. This discrepancy results in the misalignment of the modeled part. Please correct the rotation in Block 2 to ensure proper orientation and alignment.
Requirement of code generation:
1. Based on the type of error and the feedback, generate a complete correct code in the same format as the erroneous code.
Inputs: The input CAD program is: `<input_code>`
The input reference images from multiple viewpoints are: `<reference_images>` Please output in JSON format, don't output anything else. The output format is as follows:
"error type" : .., "erroneous code block ID": ..., "feedback" : ..., "correct code" : ...

Table 10: The prompt for closed-source MLLMs to perform the CAD review task.