RA-CAD: LEARNING POST-EXECUTION CRITIQUE FOR STATE-AWARE TEXT-TO-CAD GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-CAD generation translates natural-language design intent into editable and executable parametric computer-aided design (CAD) codes, reducing the expertise and effort required for manual modeling. Existing methods incorporate fixed, externally supplied, prompt-induced, or separately optimized critique mechanisms to optimize the generation process, but they do not necessarily optimize how feedback is interpreted and translated into effective corrective actions throughout the generation process. To bridge this feedback-utilization gap, we present RA-CAD (ReAct Agent for CAD), a state-aware agent that interacts with the CAD environment through a Generate–Execute–Critique–Rewrite loop. At each iteration, RA-CAD executes the current code and observes its outcome. Conditioned on the design instruction, current code, and execution feedback, the agent then generates an explicit post-execution critique as an intermediate policy action. This critique either validates the current result for termination or provides revision-oriented guidance that conditions the next rewrite. CAD Code Bootstrapping (CCB) first establishes fundamental parametric CAD coding capabilities through supervised fine-tuning. Feedback-Driven Agent Optimization (FAO) subsequently applies trajectory-level Group Relative Policy Optimization to both policy-generated code and critique sequences, assigning terminal F1 and Chamfer Distance rewards to the complete interaction trajectory. This formulation makes critique an outcome-aligned, learnable policy decision rather than an unoptimized auxiliary output. Experiments on CADFusion and Text2CAD show that RA-CAD achieves state-of-the-art execution validity and geometric quality compared with existing methods and strong proprietary language models, demonstrating the effectiveness of the proposed state-aware text-to-CAD agent.

1 INTRODUCTION

Computer-aided design (CAD) represents engineered objects through precise, editable geometric operations and is central to product design and manufacturing (Wu et al., 2021; Xu et al., 2022). Creating these models manually, however, requires substantial domain expertise and effort (Deng et al., 2023). In recent years, various automatic CAD generation technologies have been proposed to translate user specifications or design intent into executable, editable CAD representations as shown in Figure 1.(a).

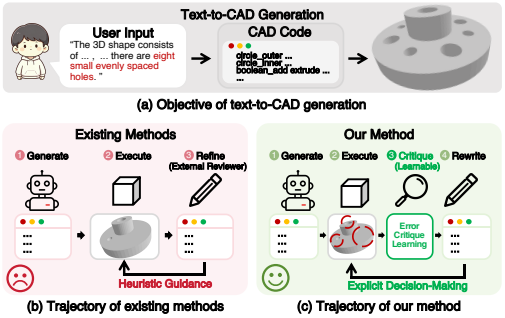


Figure 1: Comparison of CAD generation methods between existing methods and our method. Existing methods treat feedback as heuristic guidance from external visual reviewers. Our work models it as an explicit decision-making process with an agent-side component.

Early neural approaches learned unconditional distributions over CAD construction sequences (Wu et al., 2021; Xu et al., 2023). Conditional methods subsequently emerged to guide generation with natural-language descriptions, typically formulating the task as one-shot supervised prediction (Khan et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

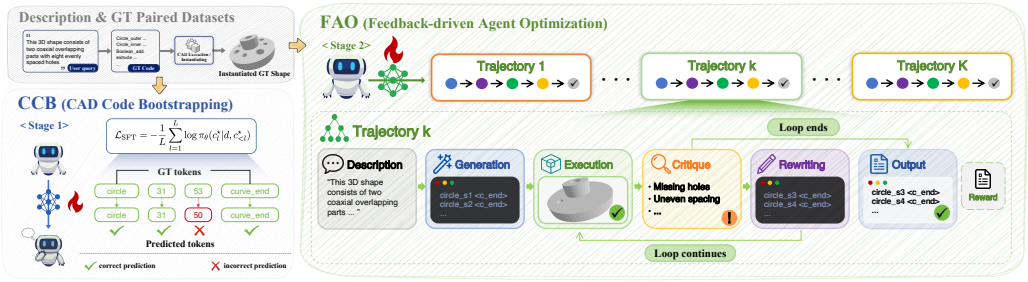


Figure 2: Overview of our training framework, including two stages: CAD Code Bootstrapping (CCB) and Feedback-driven Agent Optimization (FAO). In the CCB stage, the agent is initialized by supervised fine-tuning using paired text-CAD data to learn the syntax and geometric constraints of parametric CAD codes. In the FAO stage, the GRPO algorithm is adopted to fine-tune the complete execution trajectory of the agent, enabling the generation and critique policy to co-evolve under feedback-driven execution.

2024b; He et al., 2025; Xie & Ju, 2025; Yuan et al., 2026). Preference learning and geometric-reward optimization further improve final-output fidelity (Wang et al., 2025; Guan et al., 2025; Li et al., 2026a). Nevertheless, a complete CAD program is brittle: a single invalid parameter or inconsistent operation may prevent execution, while syntactically valid code may still instantiate the wrong geometry. A one-shot decoder cannot use these outcomes to repair its prediction.

Recent systems address this limitation through execution feedback, external visual reviewers, tool interaction, or iterative refinement (Li et al., 2026b; Gong et al., 2026; Hu et al., 2026; Fan et al., 2026; Liu et al., 2026). The availability of feedback, however, does not by itself specify how feedback utilization is represented or learned: it may serve only as a training reward, be produced by a fixed reviewer or compiler, or remain embedded in a broader reasoning trace. Regardless of the form, they treat it as heuristic guidance as shown in Figure 1.(b), and its practical value hinges on a single critical post-execution decision: judging whether the instantiated code satisfies the instruction, identifying actionable defects, and deciding whether and how to revise it. In CAD, an inaccurate decision can cause premature acceptance, unnecessary edits, or accumulated syntactic and geometric errors. Reliable generation therefore requires executable code synthesis and a post-execution critique decision that is optimized together with rewriting.

RA-CAD addresses this problem with a ReAct Agent (Yao et al., 2022) that operates as a state-aware Generate–Execute–Critique–Rewrite loop. As shown in Figure 2, RA-CAD organizes CAD generation as a critique-guided revision process under environment interaction. Rather than treating critique as a fixed post-hoc prompt or an auxiliary textual analysis step, we model it as an agent-side decision component that produces diagnostic signals for subsequent rewriting and termination. Specifically, CAD Code Bootstrapping (CCB) initializes code generation with supervised fine-tuning (SFT), whereas Feedback-Driven Agent Optimization (FAO) uses terminal code and geometry rewards to optimize all policy-generated code and critique tokens through Group Relative Policy Optimization (GRPO). RA-CAD then explicitly factorizes post-execution critique as a learnable policy component and jointly optimizes critique and code proposal/rewrite over complete environment-interaction trajectories as shown in Figure 1.(c), rather than an external reviewer as shown in Figure 1.(b). In this way, RA-CAD learns not only to propose CAD code, but also to critique intermediate results and use those critique outputs to guide later revisions. By optimizing the full interaction trajectory, the framework improves the executability, geometric accuracy, and robustness of complex CAD code generation.

The main contributions are summarized as follows:

- We formulate state-aware text-to-CAD generation as an agent interaction process with an explicitly factorized, learnable post-execution critique policy. The critique is explicitly modeled as an agent-side decision component rather than treated as a fixed prompt template or a purely post-hoc analysis step.

- We propose trajectory-level FAO with CCB, which assigns terminal CAD-quality rewards to complete interaction trajectories and jointly optimizes code generation, post-execution critique, and rewriting.
- Experimental results on CADFusion and Text2CAD demonstrate that closed-loop, critique-guided revision substantially improves execution validity and geometric quality over existing text-to-CAD methods and strong proprietary language models.

2 Related Work

2.1 CAD Generation

Sequence models such as DeepCAD (Wu et al., 2021), SkexGen (Xu et al., 2022), HNC-CAD (Xu et al., 2023), SketchGen (Para et al., 2021), and CAD-as-Language (Ganin et al., 2021) learn structured sketch or sketch-extrude representations, whereas BRepGen (Xu et al., 2024) and SolidGen (Jayaraman et al., 2023) directly model boundary representations. Methods conditioned on images/sketches (Li et al., 2022; You et al., 2025; Chen et al., 2025a; Doris et al., 2026), or point clouds (Khan et al., 2024a; Liu et al., 2024; Rukhovich et al., 2025) reconstruct CAD codes from geometric observations; FlexCAD and GeoCAD additionally support controllable editing (Zhang et al., 2025; 2026). CADReview defines CAD review as an independent task (Chen et al., 2025b), but it is aimed at error detection and code correction for existing CAD code, lacking the ability to generate CAD models from natural language. Although these inputs provide strong geometric constraints, natural language offers a more direct interface for expressing high-level design intent.

Text-to-CAD methods instead predict parametric sequences or CAD code from descriptions (Yavartanoo et al., 2024; Xie & Ju, 2025; Li et al., 2025; Govindarajan et al., 2025). Most emphasize supervised mapping, planning, or controllable generation. Their outputs may be executed for evaluation, but execution is not necessarily represented as a learned post-execution critique action within a refinement trajectory.

2.2 Feedback-Guided CAD Refinement

Feedback-guided methods differ in both the source and representation of feedback. CADFusion (Wang et al., 2025) pioneered the introduction of DPO (Rafailov et al., 2023), using rendered 3D visual feedback as preference signals to improve geometric consistency and visual fidelity. In order to further improve geometric accuracy, CAD-Coder (Guan et al., 2025) introduced GRPO (Shao et al., 2024), which further advanced geometric accuracy with GRPO, leveraging chain-of-thought (CoT) (Wei et al., 2022) inference and geometric metrics as optimization signals. ReCAD (Li et al., 2026a), PR-CAD (An et al., 2026), CME-CAD (Niu et al., 2026), ToolCAD (Gong et al., 2026), and IterCAD (Hu et al., 2026) further demonstrate the value of reinforcement learning, structured reasoning, multi-turn refinement, and environment interaction in CAD generation. Although these methods show that feedback can improve CAD generation, they primarily optimize structured inputs, generation policies, or tool-use processes. In contrast, we formulate text-to-CAD generation as a closed-loop refinement process driven by explicit learnable post-execution critique and jointly optimize the generation, rewriting, and critique policies at the trajectory level.

3 Method

3.1 Problem Formulation

Let $\mathcal{D} = \{(d^{(i)}, c^{\star(i)})\}_{i=1}^{N}$ be a text-to-CAD dataset, where d is a natural-language design description and $c^\star = (c_1^\star, \ldots, c_L^\star)$ is its ground-truth parametric CAD code. A CAD executor $\mathcal{E}$ maps a candidate code c to an execution result $e = \mathcal{E}(c)$, which records whether the code is valid and any available diagnostic message. If execution succeeds, the CAD environment instantiates geometry $\mathcal{G}(c)$. Text-to-CAD generation therefore seeks a conditional policy

$$\hat{c} \sim \pi_\theta(\cdot \mid d), \qquad \hat{e} = \mathcal{E}(\hat{c}), \tag{1}$$

where $\hat{c}$ is executable, agrees with the design intent in d, and instantiates geometry consistent with $\mathcal{G}(c^\star)$. A one-shot policy ends after producing $\hat{c}$ and cannot use $\hat{e}$ to repair the code. RA-CAD

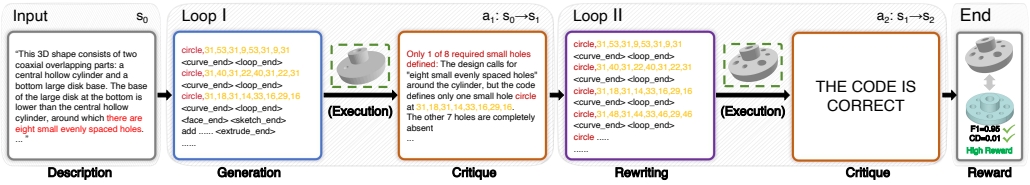


Figure 3: Overview of the agent trajectory in RA-CAD. The input description is considered the initial state s_0. At each loop, the agent executes action a_t and interacts with the CAD environment, producing a state transition from s_{t-1} to s_t.

instead formulates generation as a finite sequence of interactions between the policy and $\mathcal{E}$. Starting from a state initialized by d, round t follows the high-level transition

$$\begin{aligned} \tilde{c}_t &\sim \pi_\theta^{\mathrm{g\&r}}(\cdot \mid s_{t-1}), \quad e_t = \mathcal{E}(\tilde{c}_t), \\ \tilde{f}_t &\sim \pi_\theta^{\mathrm{crit}}(\cdot \mid \bar{s}_t), \end{aligned} \tag{2}$$

where $\tilde{c}_t$ is the initial code at $t = 1$ or a rewrite at $t > 1$, $\bar{s}_t$ is the state after execution, and $\tilde{f}_t$ is the explicit post-execution critique. The critique either accepts the current code or supplies guidance that conditions the next rewrite. The process terminates upon acceptance or at the interaction limit and returns the final code c_T. As shown in Figure 3, its complete trajectory is

$$\tau = (s_0, a_1, s_1, \ldots, a_T, s_T), \tag{3}$$

where τ represents the trajectory, s_0 represents the initial state consisting solely of d, $a_{>0}$ indicates each composite action, $s_{>0}$ represents the each complete state, and T represents the final round.

As shown in Figure 2, RA-CAD trains this policy in two stages. CAD Code Bootstrapping (CCB) first learns the description-to-code mapping from $(d, c^\star)$ pairs by supervised fine-tuning. Feedback-driven Agent Optimization (FAO) then samples complete Generate–Execute–Critique–Rewrite trajectories and applies terminal CAD-quality rewards to all policy-generated code and critique sequences. The following subsections specify these two stages and the trajectory-level objective.

3.2 CAD Code Bootstrapping

Strict CAD syntax and geometric constraints make it unreliable for reinforcement learning to learn valid CAD generation from scratch. CCB therefore first performs supervised fine-tuning on the basic language model to initialize its parametric CAD code generation capability.

During the training process, we adopt a teacher forcing policy to gradually predict each token in the target CAD code through autoregression (next-token prediction). In the l-th generation step, the model predicts the current token $c_l^\star$ based on the previous $l-1$ ground-truth tokens as conditions, and uses cross-entropy loss for supervised optimization. The entire training objective is defined as

$$\mathcal{L}_{\mathrm{SFT}} = -\frac{1}{L}\sum_{l=1}^{L} \log \pi_\theta(c_l^\star | d, c_{<l}^\star), \tag{4}$$

where L represents the length of the ground-truth code, π_θ denotes the model policy with parameters θ.

3.3 Feedback-driven Agent Optimization

While CAD Code Bootstrapping (CCB) enables the agent to generate executable CAD code, it does not explicitly optimize the agent's reasoning and decision-making ability during multi-round interaction. To address this limitation, we design a ReAct Agent that works as a closed-loop CAD generation process, and we model the process as a Markov Decision Process (MDP) (Bellman, 1957; Li et al., 2019), in which the agent makes decisions conditioned on the current design objective, historical generation result, and execution result from the environment. The complete process forms a trajectory τ, which is optimized according to the quality of the final executable CAD model, as shown in Figure 3. In this way, GRPO optimization is no longer focused on single CAD code prediction, but on the decision-making policy of the entire agent.

3.3.1 STATE

To capture both the current CAD generation result and the execution result returned by the environment, we define the state at round t as

$$s_t = (d, c_t, e_t, f_t), \tag{5}$$

where d denotes the user-provided text description, c_t denotes the current CAD code candidate, e_t denotes the results returned by the CAD execution environment, and f_t denotes the critique result associated with the current round. Therefore, each decision made by the agent is conditioned on the design objective, the current code state, the execution result, and the critique feedback.

3.3.2 ACTION

Our framework contains four functional modules: Generation, Execution, Critique, and Rewriting. Among them, Generation, Critique, and Rewriting constitute the agent-side decision process, while Execution belongs to the environment and is responsible for executing the current CAD code and returning feedback.

Instead of treating Generation, Critique, and Rewriting as isolated actions, we formulate the agent's decision at each round as a composite action:

$$a_t = (\tilde{c}_t, \tilde{f}_t), \tag{6}$$

where a_t is temporally ordered composite decision, $\tilde{c}_t$ denotes the CAD code proposed by the agent at the current round which depends on the action mode of $m_t \in \{\texttt{generate}, \texttt{rewrite}\}$, and $\tilde{f}_t$ denotes the critique output, which is the evaluation of whether the current code and its execution result meet the design objectives. Specifically, when $m_1 = \texttt{generate}$, the agent only generates the initial CAD code based on the input description. When $m_{>1} = \texttt{rewrite}$, the agent will modify the previous CAD code based on the text description, execution results, and previous evaluation records. After getting the generated code in the above steps and then obtaining the execution result, the agent critiques it again. On this basis, we can define the policy in detail as

$$\pi_\theta(a_t \mid s_{t-1}) = \pi_\theta^{\text{g\&r}}(\tilde{c}_t \mid s_{t-1})\pi_\theta^{\text{crit}}(\tilde{f}_t \mid \bar{s}_t). \tag{7}$$

where $\pi_\theta^{\text{g\&r}}$ and π_θ^{crit} denote the conditional distributions for Generation/Rewriting and Critique, respectively. After $\tilde{c}_t$ is produced, the CAD environment updates to intermediate state $\bar{s}_t$, which has been updated to $\tilde{c}_t$ and e_t, respectively. The Critique module then produces $\tilde{f}_t$ conditioned on $\bar{s}_t$. As shown in Figure 3, after fully executing action a_t, the environment state transitions from s_{t-1} to s_t.

3.3.3 REWARD

Since the ultimate goal of CAD codes is to generate parameterized models that meet design requirements and can be executed correctly, we calculate the final reward value for the complete trajectory based on the final generated code results. For the final generated result, we comprehensively consider the following aspects: how close the generated sequence is to the ground-truth sequence; whether the CAD code can be successfully executed; and geometric similarity between the generated model and ground-truth model. Therefore, the reward function can be expressed as

$$R = \lambda_1 R_{\text{F1}} + \lambda_2 R_{\text{CD}}, \tag{8}$$

where λ_1, λ_2 are weight coefficients; R_{F1} is the F1 score reward of the basic sketch topological tokens, which measures the similarity between the generated parameter sequence and the real parameter sequence; R_{CD} is the distance of the instantiated model of the final result of the trajectory, which not only measures whether the code is successfully executed, but also measures the quality of the executable model generation. Specifically, R_{CD} and R_{F1} are defined as:

$$R_{\text{CD}} = \begin{cases} e^{-\gamma \text{CD}}, & \text{if CAD code is executable} \\ 0, & \text{otherwise} \end{cases}, \tag{9}$$

$$R_{\text{F1}} = R_{\text{Avg F1}} = (\text{F1}_{\text{line}} + \text{F1}_{\text{arc}} + \text{F1}_{\text{circle}})/3, \tag{10}$$

where γ is hyperparameter and CD is the chamfer distance between the point cloud corresponding to the generated CAD model $\mathcal{G}(c)$ and the point cloud corresponding to the ground-truth CAD model $\mathcal{G}(c^\star)$, and R_{F1} only considers the basic sketch topological tokens $\{\texttt{line}, \texttt{arc}, \texttt{circle}\}$ as they dominate CAD sequences and determine structural correctness. Extrusion tokens $\{\texttt{add}, \texttt{cut}, \texttt{intersect}\}$ exhibit substantially higher numerical variability than sketch topology tokens, making token-level matching less reliable, with their quality indirectly measured by the CD reward. Due to the dependence of rewards on the entire agent inference process, they cannot be decomposed into a single step, but rather serve as the final reward for the entire trajectory. Although the reward is defined on the final trajectory outcome, the critique behavior is implicitly optimized because critique outputs constitute part of the action trajectory updated by GRPO.

3.3.4 Policy Optimization with GRPO

Given the trajectory-level reward R, we adopt GRPO to optimize the agent policy π_θ. It normalizes rewards within each group of trajectories sampled from the same input, providing a stable and computation-efficient advantage estimation. Firstly, it calculates the group-relative advantage $A_k[p]$ of the k-th trajectory of the i-th sample through the reward value, which is defined as follows

$$A_k[p] = \frac{R_k - \mu_i}{\sigma_i}, \tag{11}$$

where R_k is the final reward of the k-th trajectory, μ_i represents group-wise mean, σ_i represents standard deviation, and p is the valid token position in the trajectory. Given the advantages, we compute the token-level probability ratio

$$\rho_k[p] = \frac{\pi_\theta(l_{k,t}[p] \mid s_{k,t},\ l_{k,t}[:p])}{\pi_{\theta_{\text{old}}}(l_{k,t}[p] \mid s_{k,t},\ l_{k,t}[:p])} \tag{12}$$

between the current policy π_θ and the old policy $\pi_{\theta_{\text{old}}}$, where $l_{k,t} \in \{l_{k,1}, \ldots, l_{k,T}\}$ represents the output sequence in the t-th sub-action round, including the generated code sequence and the critique sequence. Then we optimize the clipped surrogate objective defined as

$$\begin{aligned}\mathcal{L}_{\text{GRPO}} = & -\frac{1}{K}\sum_{k=1}^{K}\frac{1}{L_k}\sum_{p=0}^{L_k-1}\min\left(\rho_k[p]\cdot A_k[p],\ \text{clip}(\rho_k[p],\ 1-\varepsilon_-,\ 1+\varepsilon_+)\cdot A_k[p]\right) \\ & +\beta\cdot\mathbb{E}_{k,p}\big(\log\frac{\pi_\theta}{\pi_{\text{ref}}} - 1 + \frac{\pi_{\text{ref}}}{\pi_\theta}\big),\end{aligned} \tag{13}$$

where K is the total number of trajectories, L_k is the length of the k-th trajectory, ε_- and ε_+ are the clipping hyperparameter that constrains the magnitude of policy updates to ensure training stability, and $\beta \cdot \mathbb{E}_{k,p}(\cdot)$ are KL regularization and its corresponding hyperparameter. By optimizing this objective with GRPO, the generation and critique policy within the agent are jointly and synergistically optimized under the guidance of the state.

4 Experiments

In this section, we conducted experiments on the CADFusion and Text2CAD datasets to comprehensively evaluate the effectiveness of the proposed method. Firstly, the experimental setups are introduced, including the datasets, training settings, and evaluation metrics. Subsequently, RA-CAD will be compared with existing representative methods and mainstream proprietary LLMs to verify its overall performance. After that, we analyze the contribution of each core module to the model performance through ablation experiments, further verifying the effectiveness of the proposed framework.

4.1 Setups

4.1.1 Datasets

We adopt CADFusion's parameter sequence construction rules (Wang et al., 2025) to reparameterize the CAD models in the Text2CAD and CADFusion datasets and construct a unified CAD code sequence. The text description retains the natural language annotations provided by the original

Table 1: Comparison of RA-CAD with existing methods on CADFusion and Text2CAD datasets. All indicators are multiplied by 10^2 for readability. ↑: The higher the better; ↓: The lower the better. The best performance is highlighted in bold.

(a) CADFusion Dataset

Method	Venue	Line F1↑	Arc F1↑	Circle F1↑	Avg F1↑	Ext F1↑	Avg CD↓	Med CD↓	IR↓
Text2CAD	NeurIPS'24	75.25	74.92	77.97	76.05	86.13	41.50	34.58	65.72
CADFusion	ICML'25	82.08	80.72	87.98	83.59	90.81	42.08	33.01	21.98
RA-CAD	-	**82.51**	**82.99**	**89.62**	**85.04**	**91.43**	**29.04**	**23.77**	**1.05**

(b) Text2CAD Dataset

Method	Venue	Line F1↑	Arc F1↑	Circle F1↑	Avg F1↑	Ext F1↑	Avg CD↓	Med CD↓	IR↓
Text2CAD	NeurIPS'24	68.08	68.92	75.58	70.86	78.53	42.98	35.57	71.76
CADFusion	ICML'25	75.43	**73.65**	80.99	76.69	84.02	49.78	40.96	21.85
RA-CAD	-	**75.82**	73.11	**81.32**	**76.75**	**86.58**	**33.42**	**28.97**	**1.25**

datasets, while performing data cleaning on the datasets to remove samples whose ground-truth parameter sequences cannot be instantiated, in order to ensure data quality and training stability. It should be noted that when processing the Text2CAD dataset, considering that user descriptions in actual CAD design scenarios typically focus on high-level semantics rather than the specific geometric parameters, we only used its intermediate-level text description for experimentation. The processed CADFusion and Text2CAD datasets contain approximately 20k and 60k samples, respectively, and the ratio of training, validation, and testing sets is retained at 90%-5%-5%.

4.1.2 Implementation Details

We use the open-source Meta-Llama-3-8B-Instruct as the base model. In the CCB stage, the CADFusion SFT training framework is used, and Low Rank Adaptation (LoRA) is used for efficient parameter tuning. The model is trained for 9 epochs, with a learning rate set to 1×10^{-4}, LoRA hyperparameters set to $r = 32$, $\alpha = 32$, and an optimizer using AdamW. In the FAO stage, we train based on the AgentLightning framework (Luo et al., 2025) and implement distributed training using Fully Sharded Data Parallel (FSDP). The training learning rate is set to 1×10^{-6}. Each input sample has $K = 8$ trajectories, and the training batch size is 16. GRPO hyperparameters set to $\lambda_1 = 0.1$, $\lambda_2 = 0.9$, $\gamma = 1$, $\beta = 10^{-3}$, $\varepsilon_- = 0.2$, $\varepsilon_+ = 0.3$. Max turn set to $T_{max} = 5$. All experiments were conducted on two NVIDIA GeForce RTX 4090 (48 GB) GPUs.

4.1.3 Baseline

We consider two baselines, the first of which is the transformer architecture for implementing end-to-end text-to-parametric CAD model generation in Text2CAD (Khan et al., 2024b). Due to significant differences between its model structure and the method proposed in this paper, we follow the original implementation. The second type is CADFusion's (Wang et al., 2025) LLM fine-tuning architecture based on visual signals for preference learning.

4.1.4 Metrics

We evaluated the model from two aspects: the parameter sequence and the instantiated CAD model. The quality of the parameter sequence was measured using the evaluation protocols of Text2CAD and CADFusion, based on the F1 scores of each topological operation (including line, arc, circle, and extrusion) and the average of basic sketch topological tokens. The model quality was assessed by instantiating the generated sequence into a 3D CAD model and comparing it with the real model point cloud. The geometric accuracy, executability, and distribution consistency were evaluated using average CD (Avg CD), median CD (Med CD), and invalidity ratio (IR).

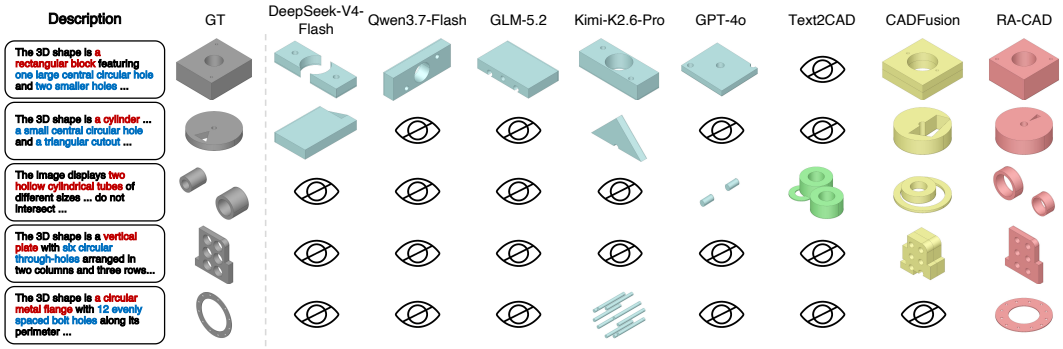


Figure 4: Qualitative comparison of baseline methods and different model variants under different training policies. Non-executable outputs are marked. The far-right column displays the overall model of our method, which has the highest executability and best preserves the geometry.

Table 2: Comparison of RA-CAD with proprietary LLMs under the same prompting protocol. The results are measured by sequence-level accuracy (Avg F1), geometric similarity (Avg CD), and execution validity (IR).

	CADFusion			Text2CAD		
Method	Avg F1↑	Avg CD↓	IR↓	Avg F1↑	Avg CD↓	IR↓
DeepSeek-V4-Flash	80.27	51.59	66.77	77.41	55.82	36.22
Qwen3.7-Flash	79.71	56.32	66.67	77.17	51.97	44.30
GLM-5.2	83.36	55.48	87.80	78.73	49.56	38.31
Kimi-K2.6-Pro	80.36	41.47	66.25	**78.78**	47.69	70.27
GPT-4o-mini	81.92	66.73	69.72	77.94	56.53	70.25
GPT-4o	81.86	61.53	54.26	78.12	50.16	63.29
RA-CAD	**85.04**	**29.04**	**1.05**	76.75	**33.42**	**1.25**

4.2 Main Results

We evaluated RA-CAD through comprehensive experiments on the CADFusion (Wang et al., 2025) and Text2CAD (Khan et al., 2024b) datasets, and compared its final parameter sequence generation quality and instantiated CAD model quality with existing methods. The final results are shown in Table 1. Our results indicate that our complete method achieved the best results on most metrics, outperforming previous work in terms of geometric accuracy. Specifically, for the CADFusion dataset, it reduced the Avg CD to 29.04 and IR to 1.05, and for the Text2CAD dataset, it reduced the Avg CD to 33.42 and IR to 1.25. Figure 4 exhibits the qualitative results. We can observe that, compared with the powerful baselines Text2CAD and CADFusion, RA-CAD has achieved significant improvements. The optimization objective of RA-CAD is more inclined towards global geometric consistency rather than simply mimicking the parameter distribution of the training set. Although CADFusion is more aggressive in predicting extrusion tokens, RA-CAD achieves a better overall performance between the final rendered geometry (CD) and invalidity ratio (IR). This indicates that RA-CAD with the learnable post-execution critique mechanism understands the geometric prior that parameter combinations determine shape, rather than merely memorizing the statistical frequencies of the parameters.

4.3 Proprietary LLMs Studies

We further evaluate its performance against several state-of-the-art proprietary LLMs. Specifically, considering that reasoning ability may have an additional impact on the generated results, to ensure comparative fairness, we adopt a uniform 8-shot prompting strategy. The quantitative results are presented in Table 2 and the qualitative results are also shown in Figure 4. Although these proprietary models exhibit remarkable capabilities in general-purpose code generation and reasoning, they perform considerably worse on parametric CAD generation. In particular, they frequently generate

Table 3: Ablation studies on the effectiveness of different components in RA-CAD on the CADFusion dataset. The experiments evaluate the contributions of CCB and FAO, among which FAO is composed of the Generation/Rewriting and Critique of the agent and GRPO with policy learning for the corresponding modules.

Variant	CCB	G&R	Crit	$\pi_\theta^{\text{g\&r}}$	π_θ^{crit}	Avg F1↑	Avg CD↓	IR↓
w/o CCB	✗	✓	✓	✓	✓	-	-	100.00
w/o FAO	✓	✗	✗	✗	✗	76.78	54.37	90.01
w/o Crit	✓	✓	✗	✓	✗	84.93	44.24	11.46
w/o GRPO	✓	✓	✓	✗	✗	81.92	38.00	18.82
fixed G&R	✓	✓	✓	✗	✓	82.19	36.97	4.42
fixed Crit	✓	✓	✓	✓	✗	83.50	32.72	7.05
Full	✓	✓	✓	✓	✓	**85.04**	**29.04**	**1.05**

syntactically invalid CAD codes or geometrically inconsistent modeling operations, resulting in a high IR.

4.4 Ablation Studies

We also isolate the impact of each component in the experiment pipeline, and the final results are shown in Table 3. We first consider that without CCB, the performance of the model would be so poor that even an effective model could not be generated, indicating the necessity of using prior knowledge in reinforcement learning based on execution results. Then, we remove the FAO stage and only retain the model trained in the CCB stage, whose limited performance indicators indicate that SFT alone cannot fully capture the spatial reasoning of complex CAD structures. Similarly, we remove the Critique from the agent and simplify the framework to traditional reinforcement learning without critique-correction generation. Since the Critique module determines whether the generated code should undergo further rewriting, ablating it directly collapses the multi-round interaction into a single-round generation process. All evaluation metrics deteriorate compared with the complete RA-CAD. These results suggest that performance gains are not only brought about by CCB and FAO themselves but also largely benefit from the post-execution critique mechanism from the agent.

To further disentangle the respective contributions of the Generation/Rewriting (G&R) and Critique modules within the agent, we conduct two additional controlled variants by fixing the policy update of one module while keeping the other trainable. Compared with the *w/o* GRPO variant, which removes reinforcement learning entirely, both *fixed* G&R and *fixed* Critique still achieve notably better performance. Notably, fixing the Critique module results in a higher Avg CD than fixing the G&R module, while fixing the G&R module yields a higher IR, implying that the two modules play asymmetric roles: the G&R module is more closely tied to execution validity, whereas the Critique module contributes more to geometric fidelity. These results confirm that jointly optimizing both modules is essential, as the co-evolution of code generation and critique provides complementary learning signals that neither module can obtain when optimized in isolation.

Overall, the complete RA-CAD achieved the best performance in all evaluation metrics. These results indicate that these components complement each other and contribute to the outstanding performance of RA-CAD.

5 Conclusion

In this paper, we presented a ReAct Agent for CAD generation (RA-CAD), which learned not only to generate CAD codes but also to critique and revise them after execution. Its central formulation factorized a complete code proposal/rewrite and an explicit post-execution critique within one state-aware policy, then jointly optimized their token sequences over complete trajectories. The reported results and ablations indicated improved geometric fidelity and validity ratio on CADFusion and Text2CAD, while leaving direct visual conditioning, per-round convergence, and broader code-space generalization for future evaluation.

AI USE STATEMENT

In this work, we used generative AI tools for aiding and polishing writing and for retrieval and discovery (e.g., finding related work). We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, implementing methods, and [assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, interpreting results, designing or providing feedback on research methodology or experiments, proposing or refining hypotheses, formulating mathematical claims, providing critical ingredients for proving mathematical claims, and assisting in the writing of proofs] are not applicable to this work. Additionally, we used generative AI tools for summarizing or analysing existing literature, sourcing/searching for information, editing a research paper to improve readability, and suggesting a structure for a research paper. We have reviewed all AI-assisted work. We sort out the research trends in the field of CAD generation and feedback-driven optimization; improve language expression and paragraph cohesion; assist in organizing the presentation logic of methods, experiments, and contributions. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several measures to ensure the reproducibility of our results. The datasets used in this work (CADFusion and Text2CAD) are publicly available, and we describe the data preprocessing and reparameterization rules in Section 4 and Appendix A and B. All model configurations, training hyperparameters (e.g., learning rate, LoRA rank, GRPO settings, reward weights), and evaluation protocols are specified in Section 4 and Appendix E. The prompts used for baselines and proprietary LLMs are provided in Appendix C. For the theoretical formulation, the MDP definition, trajectory construction, and GRPO objective are detailed in Section 3. The source code and trained model weights will be released upon paper acceptance to facilitate full reproduction.

REFERENCES

Jiyuan An, Jiachen Zhao, Fan Chen, Liner Yang, Zhenghao Liu, Hongyan Wang, Weihua An, Meishan Zhang, and Erhong Yang. Pr-cad: Progressive refinement for unified controllable and faithful text-to-cad generation with large language models, 2026. URL https://arxiv.org/abs/2604.19773.

Richard Ernest Bellman. *A Markovian decision process.* RAND Corporation, Santa Monica, CA, 1957.

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

Jiali Chen, Xusen Hei, HongFei Liu, Yuancheng Wei, Zikun Deng, Jiayuan Xie, Yi Cai, and Li Qing. CADReview: Automatically reviewing CAD programs with error detection and correction. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9909–9927, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.489. URL https://aclanthology.org/2025.acl-long.489/.

Yuanzhe Deng, James Chen, and Alison Olechowski. What sets proficient and expert users apart? results of a computer-aided design experiment. *Journal of Mechanical Design*, 146(1):011401, 10 2023. ISSN 1050-0472. doi: 10.1115/1.4063360. URL https://doi.org/10.1115/1.4063360.

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, 03 2026. ISSN 1050-0472. doi: 10.1115/1.4071305. URL https://doi.org/10.1115/1.4071305.

Fengxiao Fan, Jingzhe Ni, Xiaolong Yin, Sirui Wang, Xingyu Lu, Qiang Zou, Ruofeng Tong, Min Tang, and Peng Du. Caddesigner: Conceptual cad model generation with a general-purpose agent, 2026. URL https://arxiv.org/abs/2508.01031.

Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 5885–5897. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/2e92962c0b6996add9517e4242ea9bdc-Paper.pdf.

Yifei Gong, Xing Wu, Wenda Liu, and Tukang. TOOLCAD: Exploring tool-using large language models in text-to-CAD generation with reinforcement learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 23161–23188, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1160. URL https://aclanthology.org/2026.findings-acl.1160/.

Prashant Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. Cadmium: Fine-tuning code language models for text-driven sequential cad design. *arXiv preprint arXiv:2507.09792*, 2025.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. Cad-coder: Text-to-cad generation with chain-of-thought and geometric reward. *arXiv preprint arXiv:2505.19713*, 2025.

Changqi He, Shuhan Zhang, Liguo Zhang, and Jiajun Miao. Cad-coder: Text-guided cad files code generation, 2025. URL https://arxiv.org/abs/2505.08686.

Tao Hu, Jiaxin Ai, Licheng Wen, Xueheng Li, Shu Zou, Siqi Li, Nianchen Deng, Xinyu Cai, Hongbin Zhou, Pinlong Cai, Daocheng Fu, Yu Yang, Hairong Zhang, Botian Shi, and Xuemeng Yang. Itercad: An iterative multimodal agent for visually-grounded cad generation and editing, 2026. URL https://arxiv.org/abs/2606.13368.

Pradeep Kumar Jayaraman, Joseph George Lambourne, Nishkrit Desai, Karl Willis, Aditya Sanghi, and Nigel J. W. Morris. Solidgen: An autoregressive model for direct b-rep synthesis. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=ZR2CDgADRo. Featured Certification.

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

Mohammad Sadil Khan, Sankalp Sinha, Sheikh Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL https://openreview.net/forum?id=5k9XeHIK3L.

Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J. Mitra. Free2cad: parsing freehand drawings into cad commands. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530133. URL https://doi.org/10.1145/3528223.3530133.

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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 18563–18573, June 2025.

Jiahao Li, Yusheng Luo, Yunzhong Lou, and Xiangdong Zhou. Recad: Reinforcement learning enhanced parametric cad model generation with vision-language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(8):6190–6198, Mar. 2026a. doi: 10.1609/aaai.v40i8.37544. URL https://ojs.aaai.org/index.php/AAAI/article/view/37544.

Quan-Lin Li, Jing-Yu Ma, Rui-Na Fan, and Li Xia. An overview for markov decision processes in queues and networks, 2019. URL https://arxiv.org/abs/1907.10243.

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 *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=PzIc2TxhwN.

Yang Liu, Daxuan Ren, Yijie Ding, Jianmin Zheng, and Fang Deng. Multi-agent cad code generation. In *Proceedings of the Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers*, SIGGRAPH Conference Papers '26, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725548. doi: 10.1145/3799902.3811067. URL https://doi.org/10.1145/3799902.3811067.

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

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.03680.

Ke Niu, Haiyang Yu, Zhuofan Chen, Zhengtao Yao, Weitao Jia, Xiaodong Ge, Jingqun Tang, Benlei Cui, Bin Li, and Xiangyang Xue. Cme-cad: Heterogeneous collaborative multi-expert reinforcement learning for cad code generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 39272–39281, 2026.

Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. Sketchgen: Generating constrained cad sketches. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 5077–5088. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/28891cb4ab421830acc36b1f5fd6c91e-Paper.pdf.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-cad generation through infusing visual feedback in large language models. In *International Conference on Machine Learning*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6772–6782, October 2021.

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

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. In *International Conference on Machine Learning*, pp. 24698–24724. PMLR, 2022.

Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable cad model generation. In *International Conference on Machine Learning*, pp. 38443–38461, 2023.

Xiang Xu, Joseph Lambourne, Pradeep Jayaraman, Zhengqing Wang, Karl Willis, and Yasutaka Furukawa. Brepgen: A b-rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics (TOG)*, 43(4):1–14, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Mohsen Yavartanoo, Sangmin Hong, Reyhaneh Neshatavar, and Kyoung Mu Lee. Text2cad: Text to 3d cad generation via technical drawings, 2024. URL https://arxiv.org/abs/2411.06206.

Yang You, Mikaela Angelina Uy, Jiaqi Han, Rahul Thomas, Haotong Zhang, Yi Du, Hansheng Chen, Francis Engelmann, Suya You, and Leonidas Guibas. Img2cad: Reverse engineering 3d cad models from images through vlm-assisted conditional factorization. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, SA Conference Papers '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400721373. doi: 10.1145/3757377.3763891. URL https://doi.org/10.1145/3757377.3763891.

Bo Yuan, Zelin Zhao, Petr Molodyk, Bin Hu, and Yongxin Chen. Clarify before you draw: Proactive agents for robust text-to-cad generation, 2026. URL https://arxiv.org/abs/2602.03045.

Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. Flexcad: Unified and versatile controllable cad generation with fine-tuned large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 3204–3227, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/090b23d52bc2722eef2fbf79c5ebf9ec-Paper-Conference.pdf.

Zhanwei Zhang, Junjie Liu, Wenxiao Wang, Binbin Lin, Liang Xie, Chen Shen, Deng Cai, et al. Geocad: Local geometry-controllable cad generation with large language models. *Advances in Neural Information Processing Systems*, 38:130989–131014, 2026.

Listing 1: A representative sample of CADFusion's description structures. The three-stage structure has been highlighted in bold font.

```
"Description": "[Shape Overview] The 3D shape consists of a
    rectangular prism extending into a semi-cylinder with a curved end
    . [Shape Details] It features one large circular hole on the
    rectangular part and four smaller, equally sized holes aligned
    along the semi-cylindrical section. [Shape Applications] The shape
     is symmetrically aligned along its central axis and potentially
    designed for mounting or connecting with other components."
```

Listing 2: A representative sample of CADFusion's description structures. It can be seen that the descriptions from L0 to L3 become increasingly detailed. The description of L3 is quite detailed, involving multiple curves and instructions for Boolean operations. Due to the excessive amount of content, it has been omitted for the time being.

```
{
  "L0": "A rectangular prism with a cylindrical hole in the center.
     The prism has four circular cutouts on its top surface."
  "L1": "The CAD model features a rectangular bar with a cylindrical
     hole in the center and four circular cutout sections on top."
  "L2": "The CAD model consists of a rectangular prism with a
     cylindrical hole in the center and circular cutouts on the top
     surface. This part is created by sketching inside a coordinate
     system and extruding and cutting shapes in 3D. The resulting
     part has a height of approximately 0.2 units."
  "L3": "The first part of the CAD model is a rectangular prism with a
      cylindrical hole in the center and four circular cutouts on the
      top surface. Begin by creating a new coordinate system with
     Euler angles of [0.0, 0.0, 0.0] and a translation vector of
     [0.0, 0.0, 0.0]. In the first 2D sketch, draw a rectangle by
     creating four lines with the following endpoints:* Line 1: [0.0,
      0.0] and [0.3, 0.0] * Line 2: [0.3, 0.0] and [0.3, 0.3] ......"
}
```

A DESCRIPTION

In the text-to-CAD generation task, the format and granularity of the input text description have a significant impact on the learning efficiency and generation quality of the model. The description structures of the two datasets we selected follow the following rules.

CADFusion (Wang et al., 2025) adopts a three-stage structured description paradigm, dividing the text expression of the CAD model into three levels: **(1) Shape Overview**, which is used to describe the overall geometric configuration and macroscopic features; **(2) Shape Details**, covering specific geometric attributes such as size ratios, component quantities, and spatial arrangement; **(3) Shape Applications**, which clarify the potential functions or usage scenarios of this shape. As shown in Listing 1, Shape Overview is a mandatory part, while the details and applications parts can be selected flexibly according to the complexity of the model.

Text2CAD (Khan et al., 2024b) establishes a four-level progressive description system, as shown in Listing 2, dividing the information granularity and professional level from simple to complex into four grades:

- **L0 (Abstract Level):** It automatically describes the overall shape of the CAD model from a visual perspective, focusing on category generalization.
- **L1 (Beginner Level):** It simplifies the description of design steps using non-professional terms, avoiding complex parameters and measurement information, suitable for preliminary design or non-professional users.
- **L2 (Intermediate Level):** While maintaining readability, introduces generalized descriptions of geometric shapes, balancing technical accuracy and expression universality.

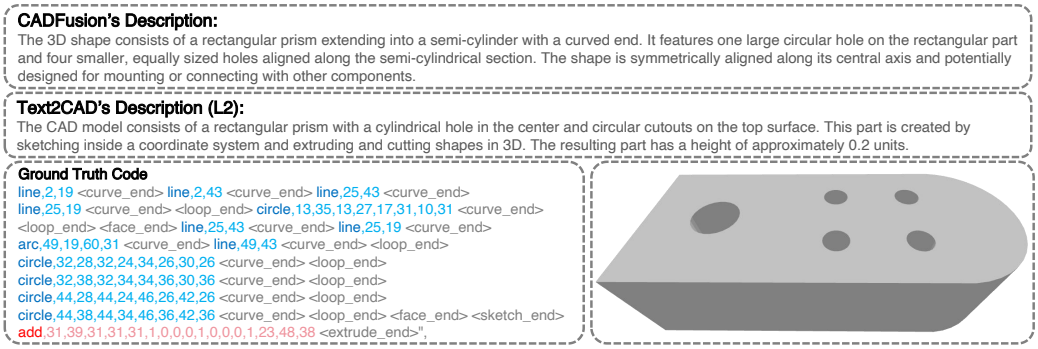


Figure 5: A representative sample of CAD code and its corresponding descriptions. The upper and middle areas respectively represent the descriptions of CADFusion and Text2CAD. The lower left area represents the CAD code sequence, where all types of tokens are distinguished by different colors. All of them correspond to the same CAD model, which is displayed in the lower right corner.

- **L3 (Expert Level):** Provides precise geometric parameters and relative measurement information, including numerical descriptions such as coordinates, dimensions, and directions, meeting the refined modeling needs of professional designers.

Considering that user descriptions in actual CAD design scenarios typically focus on high-level semantics such as part functionality, structural features, and overall geometric attributes, it is difficult to accurately specify the specific geometric parameters corresponding to each edge and arc. Moreover, due to the different construction of CAD code sequences, when converting the Text2CAD dataset, we mainly consider the L2 level description.

B CAD CODE

To achieve the structured modeling and editability of CAD models, this paper adopts a serialized representation format based on "Sketch-and-Extrude". This sequence rule is derived from SkexGen (Xu et al., 2022) and is further standardized by CADFusion into a unified textual CAD language. Each CAD instance is composed of several sketches (Sketch) and extrusion operations in the order of design history. The specific coding rules are as follows:

- **Topological Tokens:** Topological Tokens are used to identify the basic geometric elements in 2D sketches, including three categories: $\{\texttt{line}, \texttt{arc}, \texttt{circle}\}$, representing the line primitive, arc primitive, and circle primitive, respectively.
- **Topological Parameter Tokens:** Each type of Topological Token must be followed by a different number of coordinate parameters, in the following format: (1) $\texttt{line}, x_0, y_0$; (2) $\texttt{arc}, x_0, y_0, x_1, y_1$; (3) $\texttt{circle}, x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3$. Among these, x and y represent the horizontal and vertical coordinates, respectively, and (x_0, y_0) represents the starting point coordinates of the curve.
- **Extrusion Tokens:** Extrusion Tokens are used to define the types of volume operations in 3D modeling, including three types of Boolean operations: $\{\texttt{add}, \texttt{cut}, \texttt{intersect}\}$, representing adding the new entity to the existing entity (which can be empty), cutting the corresponding entity from the existing entity, and intersecting the new entity with the existing entity, respectively.
- **Extrusion Parameter Tokens:** Each type of Extrusion Token must be followed by a total of 17 numerical parameters, which are organized in order as follows: (1) Parameters 1–2: ext_v (extrusion height on the upper and lower planes); (2) Parameters 3-5: ext_T (3D extrusion translation vector); (3) Parameters 6-14: ext_R (3*3 rotation matrix); (4) Parameter 15: $scale_quan$ (quantized value of the normalized scale); (5) Parameters 16-17: $offset_quan$ (quantized value of the center offset).
- **Hierarchical End Tokens:** To clearly define the multi-level nested boundaries of the CAD structure, the following Hierarchical End Tokens are defined: {<curve_end>,

Listing 3: Prompt of the SFT training format. The model receives the full description and partial code sequence as input, while the loss is computed only on the next tokens (the part after the prompt).

```
Below is a description of a 3D shape:
{Description}
Generate a Computer-Aided Design(CAD) command sequence of the 3D shape
    :
{Ground_Truth_Code[:i]}
```

Listing 4: Prompt of the Generation module.

```
"system":"
  You are a professional CAD code generation expert, using token
      sequences to represent CAD models.
  Based on the user's description of the CAD model, directly generate
      a valid token sequence. Strictly adhere to the continuous
      contour and hierarchical structure.
  ",
"user":"
  Below is a description of a 3D shape:
  {Description}
  Generate a Computer-Aided Design(CAD) command sequence of the 3D
      shape:
  "
```

<loop_end>, <face_end>, <sketch_end>, <extrude_end>}, marking the end of the current curve, loop, face, sketch, and extrusion, respectively.

Among them, each Topological Parameter Token only defines the starting point of the Topological Token and the intermediate control points along the path, while the coordinate of the endpoint is implicitly determined by the starting point of the next curve. When it is necessary to close the current loop, the endpoint of the last curve points back to the starting point of the first curve in the loop, thus achieving the closure of the loop. For sketches, each sketch is composed of multiple faces, and each face contains one or more loops. The first loop defines the external boundary, and subsequent loops define the internal holes. This hierarchical constraint ensures that the generated CAD sequence is geometrically and topologically legally closed. All coordinate parameters and continuous parameters are uniformly quantized into 6-bit discrete Tokens (with a total of 64 possible values), enabling the entire CAD sequence to be input into the LLM in a unified text Token form for processing.

C PROMPT

C.1 CCB PROMPT

As the first training stage of RA-CAD, CCB initializes the agent's parametric CAD coding ability through supervised fine-tuning. The SFT prompt format is shown in Listing 3. Each training sample is an instruction-style prompt that first presents the natural-language description of the target shape and then requests the corresponding CAD command sequence. To train the model in a next-token prediction manner, the prompt is constructed with only the first i tokens of the ground-truth code (Ground_Truth_Code[:i]), and the cross-entropy loss is computed exclusively on the subsequent tokens that follow the prompt. This formulation teaches the model to continue a partially generated CAD sequence rather than to re-predict the prefix it has already observed. At the same time, all training samples share the same description format and the same token-level CAD code used in the subsequent FAO stage.

C.2 FAO PROMPT

During the FAO stage, the agent is organized as a Generate–Execute–Critique–Rewrite loop, in which each module is driven by a dedicated role-specific prompt. The Generation module, as shown in Listing 4, is instructed as a professional CAD code generation expert and is required to directly

Listing 5: Prompt of the Rewriting module.

```
"system":"
  You are a professional CAD code regeneration expert, using token
      sequences to represent CAD models.
  Rewrite the CAD code sequence based on the feedback.
  Please fix the issues in the code and generate a correct compact
      format sequence.
  ",
"user":"
  Below is a description of a 3D shape:
  {Description}
  Original code:
  {Code}
  Execution result:
  {Execution}
  Feedback:
  {Feedback}
  Please regenerate a Computer-Aided Design(CAD) command sequence of
      the 3D shape:
  "
```

Listing 6: Execution module. The execution module returns either a success message with structural validation statistics, or a failure message containing the exact error location and description.

```
Input:
{CAD_Code}

Output - Success:
  Code executed successfully. The generated model is structurally
      sound.
  Validation passed - {L} token(s), {I} sketch/extrude pair(s),
  {J} face(s), {K} loop(s), {H} curve(s)

Output - Failure:
  Code execution failed with error: Cannot be parsed or rendered into
      an actual 3D model.
  Found {N} error(s):
  1.Sketch-Extrude {i} Face {j} Loop {k} Curve {h}: {specific error}
  2. ...
```

produce a valid token sequence while strictly adhering to the continuous contour and hierarchical structure of the CAD language. The Rewriting module, as shown in Listing 5, is prompted as a professional CAD code regeneration expert and takes the description, the original code, the execution result, and the critique feedback as input, with the instruction to fix the identified issues and regenerate a correct and compact sequence. The Execution module acts as the bridge between the agent and the CAD environment. As shown in Listing 6, it receives the CAD code generated or rewritten by the Generation/Rewriting modules, parses the token sequence, validates the hierarchical structure, and instantiates the corresponding 3D model. When execution succeeds, it returns a success message together with the structural validation statistics, including the total number of tokens, sketch/extrude pairs, faces, loops, and curves. When execution fails, it returns a failure message and reports the specific error location, such as the affected Sketch-Extrude, Face, Loop, and Curve indices, so that the agent can identify exactly where the problem occurred. The Critique module receives the CAD description, the generated code, and its execution result. As shown in Listing 7, its system prompt enumerates the topological tokens, extrusion tokens, and hierarchical end tokens of the CAD representation, and asks the model to review the code item by item along three dimensions: program executability, design intent consistency, and geometric reasonability. Its output format requires the model to first perform validation reasoning and then return a structured verdict, which standardizes the feedback consumed by the subsequent module. The same prompts are reused consistently throughout FAO training and inference. During GRPO, each complete interaction trajectory is produced by rolling out these four modules in turn against the real CAD execution environment, and all policy-generated code and critique tokens are jointly optimized under terminal F1 and Chamfer

Listing 7: Prompt of the Critique module.

```
"system":"
  You are a CAD code critique expert who pays great attention to
      details of the following tokens that represent CAD models.
  Token types:
  1) Topological tokens: line, arc, circle
  2) Extrusion tokens: add, cut, intersect
  3) Hierarchical end tokens: <curve_end>, <loop_end>, <face_end>, <
      sketch_end>, <extrude_end>
  Please review the current CAD code item by item from the following
      three dimensions:
  1. Program Executability
  -Can the code be compiled and run without errors through the CAD
      execution environment?
  -If the execution fails, what is the specific error type and on
      which line/parameter did the error occur?
  2. Design Intent Consistency
  -Does the generated CAD model fully cover all geometric features and
       functional requirements in the design description?
  -Are the features (stretch, rotation, hole, chamfer, array, etc.)
      appearing in the correct position and size?
  -Are there any missing or redundant features that do not conform to
      the design description?
  3. Geometric Reasonability
  -Is the overall scale of the model coordinated? Is there a clearly
      unreasonable aspect ratio or wall thickness?
  -Are there any unexpected intersections, penetrations, or gaps
      between geometric entities?
  -Is there degenerate geometry (zero volume, coplanarity, self
      intersection, etc.)?
  ### Output Format ###
  You always first perform your validation reasoning by wrapping your
      analysis in <think> and </think>. In your reasoning, examine the
       code item by item against the specification and the execution
      result.
  After completing your reasoning, provide your final validation
      result within <result> and </result> tags:
  - If errors are found: A bulleted list of specific issues, each on a
       new line.
  - If no errors are found: The exact phrase "THE CODE IS CORRECT".
  ",
"user":"
  CAD description:
  {Description}
  Generated code:
  {Code}
  Execution result:
  {Execution}
  "
```

Distance rewards. The role prompts thus provide a stable interface between the environment, the critique decision, and the rewrite, ensuring that the policy learns to turn execution feedback into explicit corrective actions rather than relying on fixed or unoptimized textual guidance.

C.3 Few-shot Prompt

To evaluate RA-CAD against strong proprietary language models under a fair and reproducible protocol, we use the few-shot prompt shown in Listing 8 for all closed-source baselines. The prompt first states the task of generating a CAD command sequence from a shape description and briefly explains the structure of the sequence, namely sketches composed of lines, arcs, and circles together with the extrusions that turn a sketch into a 3D volume. It then provides eight complete description-code demonstration pairs and explicitly indicates that the tokenized values range from 0 to 63, which

Listing 8: Prompt of the few-shot. For different datasets, we respectively selected 8 distinct samples as the few-shot examples.

```
Below is a description of a 3D shape:
{Description}
Generate a Computer-Aided Design (CAD) command sequence of the 3D shape. The command sequence involves sketches such as lines, arcs, and circles, each marked by the endpoints, and extrusions that make the sketch into 3D volumes.
Here are some examples and their value range is 0-63
1. Description: {Description[0]}, CAD Command Sequence: {Ground_Truth_Code[0]}
2. Description: {Description[1]}, CAD Command Sequence: {Ground_Truth_Code[1]}
3. Description: {Description[2]}, CAD Command Sequence: {Ground_Truth_Code[2]}
4. Description: {Description[3]}, CAD Command Sequence: {Ground_Truth_Code[3]}
5. Description: {Description[4]}, CAD Command Sequence: {Ground_Truth_Code[4]}
6. Description: {Description[5]}, CAD Command Sequence: {Ground_Truth_Code[5]}
7. Description: {Description[6]}, CAD Command Sequence: {Ground_Truth_Code[6]}
8. Description: {Description[7]}, CAD Command Sequence: {Ground_Truth_Code[7]}
Now it's your turn. Remind that this is your description: {Description}.
No explanation is needed. Only return your final sequence, and in one line.
```

corresponds to the 6-bit quantization of all coordinate and continuous parameters. Finally, the model is asked to generate the sequence for the target description without providing any explanation and to return only the final sequence in a single line.

Considering that there are differences in the description methods between the CADFusion and Text2CAD datasets, we selected 8 representative samples from each of the two datasets as the few-shot examples, as shown in Table 4 and Table 5 respectively. When selecting, we took into account the difficulty gradient and structural complexity of the samples, and ensured that the examples covered as many diverse types of CAD tokens as possible, thereby more comprehensively reflecting the generation characteristics of the corresponding datasets.

D Module Details

The RA-CAD framework contains four functional modules: Generation, Execution, Critique, and Rewriting. The four modules are described as follows:

1. Generation The Generation module is responsible for producing the initial CAD code from the input text description. Given an initial state s_0 that only contains d, the Generation module will generate the initial candidate CAD code c_1 and pass it into the environment. Rather than being directly treated as the final output, the generated code is regarded as an intermediate hypothesis for the current design objective, which will be further verified and refined through interaction with the execution environment.

2. Execution The Execution module belongs to the environment rather than the agent itself. It instantiates and executes the CAD code c_t proposed by the agent, detects syntax errors, invalid parameters, and other execution failures, and returns execution status together with the corresponding error message, if it exists, to the environment. Then it forms an intermediate state $\bar{s}_t$.

3. Critique The Critique module evaluates the current CAD result by jointly considering the intermediate state $\bar{s}_t$ (including input description d, the generated CAD code c_t, and the execution result e_t). It determines whether the current code satisfies the design requirements in terms of intent con-

ID	CAD Model Structure	Topological Tokens	Extrusion Tokens
0002/00027926	Rectangular prism	**4** line × 4	**1** add × 1
0034/00343336	Triangular prism	**3** line × 3	**1** add × 1
0030/00302523	Rectangular prism + 2 cylindrical holes	**6** line × 4, circle × 2	**1** add × 1
0002/00023779	Disc + Center boss + 5 cylindrical holes	**23** circle × 13, arc × 10	**2** add × 2
0021/00218782	Rectangular prism + Semi-cylindrical top + Cylindrical hole	**6** line × 4, arc × 1, circle × 1	**1** add × 1
0034/00344524	Rectangular prism + Cutout square hole	**8** line × 8	**2** add × 1, cut × 1
0027/00272814	L-shaped prism + Inner corner rounded	**5** line × 4, arc × 1	**1** add × 1
0025/00250827	Rectangular prism + Center square hole + Center boss + Center circular hole	**13** line × 12, circle × 1	**2** add × 2

Table 4: Few-shot samples from the CADFusion dataset. We used the descriptions and true values of these eight samples as the input for the closed-source model for its reasoning.

ID	CAD Model Structure	Topological Tokens	Extrusion Tokens
0000/00000894	Rectangular prism	**4** line × 4	**1** add × 1
0000/00001260	Cylinder	**1** circle × 1	**1** add × 1
0001/00016772	Rectangular prism + Rounded corners + Curved contour	**9** line × 5, arc × 4	**1** add × 1
0001/00019015	Rectangular prism + Cylinder	**8** line × 8	**2** add × 2
0000/00002694	Rectangular prism + Central circular hole + Cutout circular pit + Rectangular prism	**13** line × 8, circle × 5	**3** add × 2, cut × 1
0055/00552776	Rectangular prism + Central square hole + Cylinder	**12** line × 12	**2** add × 2
0057/00578900	T-shaped prism + 5 cylindrical holes	**15** line × 10, circle × 5	**1** add × 1
0055/00558114	Complex multi-component assembly	**18** line × 10, arc × 3, circle × 5	**4** add × 4

Table 5: Few-shot samples from the Text2CAD dataset. We used the descriptions and true values of these eight samples as the input for the closed-source model for its reasoning.

sistency, geometric validity, and overall model quality. The critique output not only decides whether further optimization is needed, but also provides revision suggestions for the next round.

4. Rewriting When the Critique results indicate that the current code still has room for improvement, the rewriting module will work as the start of a new round of actions for the agent. It will rewrite the

code based on the complete state $s_t = (d, c_t, e_t, f_t)$ updated from the previous round of actions. The modified code c_{t+1} will then be sent back to the Execution module for verification. The interaction loop continues until the critique module determines that the design goal has been satisfied or the maximum number of iterations is reached.

E Implementation Details

The FAO stage is implemented with the verl framework on a single node with two GPUs, and the agent is initialized from Meta-Llama-3-8B-Instruct. Rollouts are generated with the asynchronous vLLM engine at a temperature of 0.7, sampling $n = 8$ candidate trajectories per prompt; prompts and per-turn responses are both capped at 1,024 tokens, respectively, and truncation is disabled so that any over-length sequence raises an error instead of being silently cut. Since the agent operates as a Generate–Execute–Critique–Rewrite loop, multi-turn rollouts use the Hermes tool-calling format, which allows each module's structured output to be parsed and routed through the CAD execution environment. Each training step samples 16 prompts (128 trajectories in total), with a mini-batch size of 8 and a micro-batch size of 4 per GPU. The policy and reference models are trained with FSDP and parameter offloading; gradient checkpointing is enabled, and the reference model is kept frozen for computing reference log probabilities. GRPO is applied with group-normalized advantages, a fixed KL penalty with coefficient 0.001 incorporated into the reward, policy clipping ratios of 0.2 and 0.3, and a learning rate of 10^{-6}, for a single epoch over the training prompts.

As for the assessment, generated CAD sequences are first parsed into sketch-and-extrude operations using the same 6-bit quantization as training, where each quantized value q is dequantized as

$$\hat{x} = q \cdot (\max - \min)/(2^6 - 1) + \min,$$

with sketch coordinates in $[-1, 1]$, extrusion heights and translations in $[-1, 1]$, scale in $[0, 1.4]$, and center offset in $[-0.9, 0.9]$. A sequence is counted as valid only if it parses into a consistent sequence of sketches and extrusions, passes the structural checks of the parser (correct token counts, closed loops, non-degenerate curves, and valid Boolean operations), and can be rendered through the CAD execution environment. The invalidity ratio is computed as

$$IR = (1 - N_{\text{valid}}/N_{\text{total}}) \times 100\%.$$

For geometric evaluation, 2,000 points are sampled from the surface of each rendered mesh (downsampled from denser point clouds when necessary) and normalized by the maximum absolute coordinate. The Chamfer distance between a generated point cloud P and its ground-truth point cloud Q is defined as

$$CD(P, Q) = \frac{1}{|P|} \sum_{p \in P} \min_{q \in Q} \|p - q\|_2^2 + \frac{1}{|Q|} \sum_{q \in Q} \min_{p \in P} \|p - q\|_2^2$$

and the reported Avg CD is the mean of the diagonal entries of the pairwise distance matrix, i.e., each generated model is compared only with its corresponding ground-truth model. For sequence-level accuracy, F1 scores are computed per primitive type (line, arc, and circle) within each sketch-extrude segment: for type t, with occurrence counts c_t^{gt} and c_t^{pred},

$$m_t = \min(c_t^{\text{gt}}, c_t^{\text{pred}}),$$

$$\text{Precision}_t = m_t / c_t^{\text{pred}},$$

$$\text{Recall}_t = m_t / c_t^{\text{gt}},$$

$$F1_t = 2 \cdot \text{Precision}_t \cdot \text{Recall}_t / (\text{Precision}_t + \text{Recall}_t),$$

where per-type scores are averaged over the evaluated samples, and Avg F1 is the mean of the line, arc, and circle F1 scores. The same pipeline additionally computes the Jensen–Shannon divergence (JSD) between the per-cell occupancy probabilities of the generated and ground-truth point clouds on a $28 \times 28 \times 28$ grid in the unit cube.

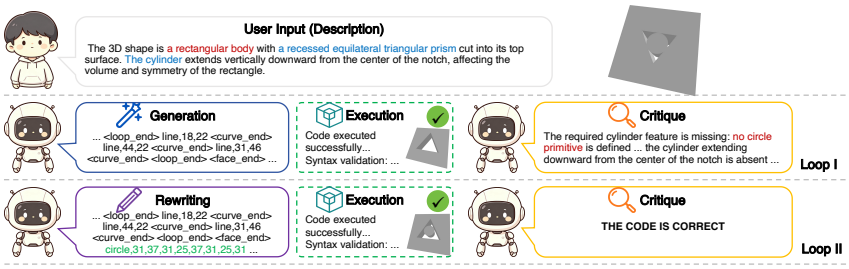


Figure 6: The text-to-CAD generation process case. The first loop model is missing a cylinder, while the second loop adds the circle feature for correction.

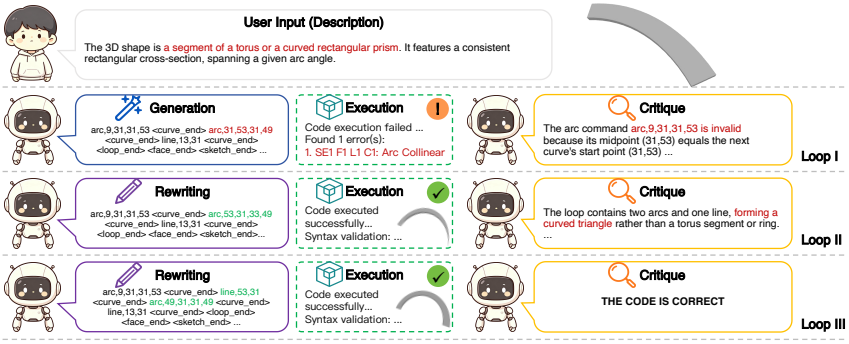


Figure 7: The text-to-CAD generation process case. The first loop fails execution due to an arc with duplicate points, the second loop executes but forms a curved triangle rather than a ring, and the third loop correctly builds the shape from two concentric arcs and two straight sides.

F Case Studies

We further provide qualitative case studies to illustrate how RA-CAD improves CAD code in the closed-loop Generate–Execute–Critique–Rewrite paradigm using execution feedback and explicit critique. Figure 6 and Figure 7 show the text-to-CAD generation process: the first process shows how RA-CAD corrects a missing structural feature through feedback-driven rewriting, while the second process shows how RA-CAD leverages the detailed failure messages returned by the CAD execution environment to locate and repair errors.

Figure 8 further presents qualitative comparisons of the final outputs on several representative cases. Compared with strong proprietary models (DeepSeek-V4-Flash, Qwen3.7-Flash, GLM-5.2, Kimi-K2.6-Pro, and GPT-4o) and existing methods (Text2CAD and CADFusion), RA-CAD best preserves the requested structures and geometric relations while remaining executable, whereas most baselines produce non-executable or geometrically inconsistent outputs.

G Comparison Protocol and Reproducibility Constraints

Several recent text-to-CAD and agent-based methods, including CAD-coder (Guan et al., 2025), PR-CAD (An et al., 2026), ReCAD (Li et al., 2026a), iterCAD (Hu et al., 2026), ToolCAD (Gong et al., 2026), Seek-CAD (Li et al., 2026b), CME-CAD (Niu et al., 2026), and CAD-Factory (Liu et al., 2026), are not publicly reproducible at the time of submission: their source code, trained checkpoints, or complete evaluation pipelines are unavailable. In addition, some of them use different CAD representations (e.g., B-Rep, mesh, or alternative parametric DSLs), input modalities (e.g., images, sketches, or point clouds), datasets, or evaluation protocols, which prevents a faithful apples-to-apples comparison under our unified sketch-and-extrude code representation and terminal CAD-quality metrics.

To ensure that all reported comparisons are based on identical data preprocessing, execution environments, and evaluation metrics, we therefore compare against methods for which implementations

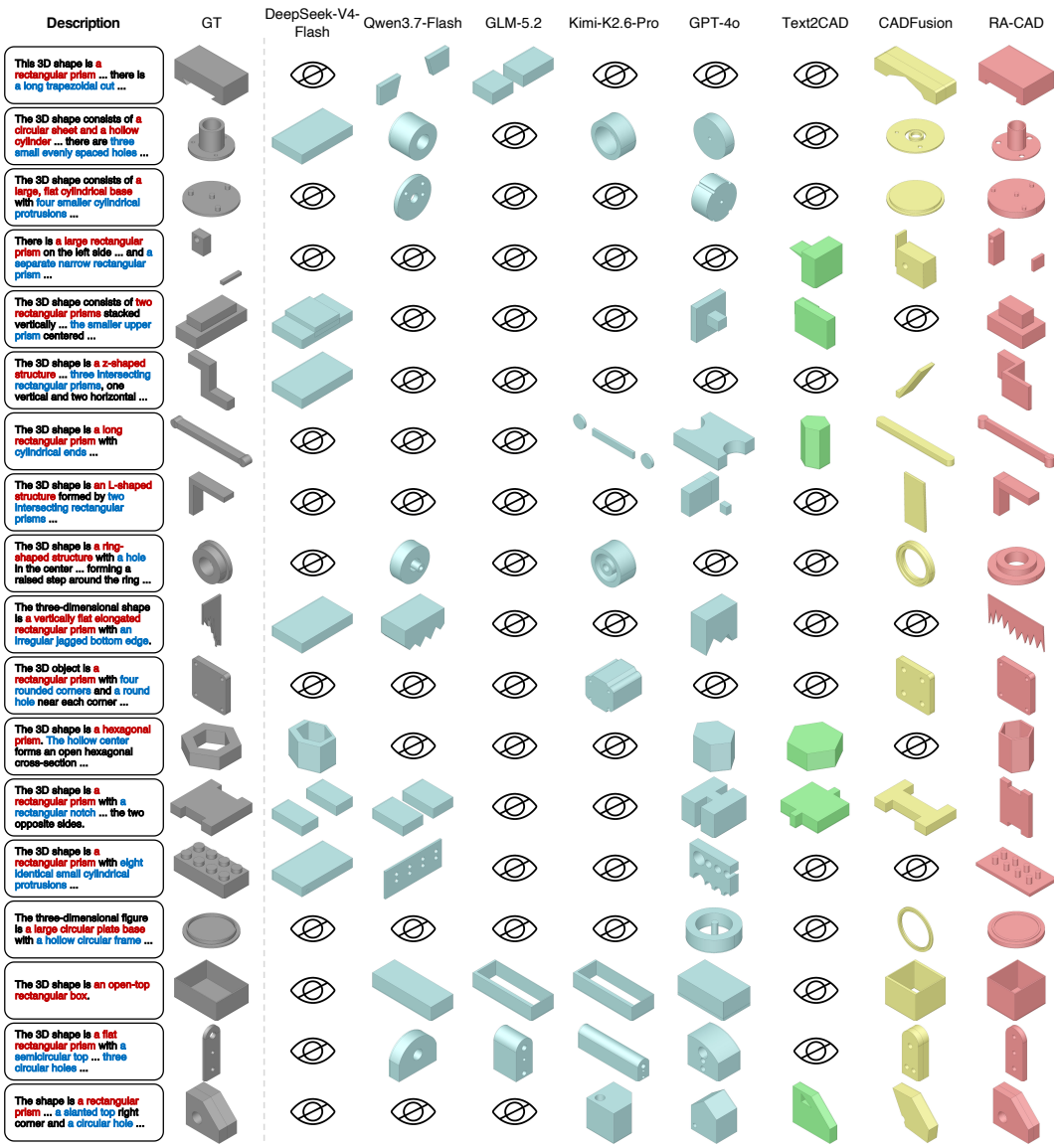


Figure 8: Qualitative comparison of baseline methods and different LLM variants under different training policies. Non-executable outputs are marked.

and evaluation protocols are publicly accessible, including Text2CAD and CADFusion. We further include strong proprietary LLMs under a fixed 8-shot prompting protocol as an additional reference point. This comparison protocol does not imply universal superiority over methods that could not be reproduced under the same setting; rather, it isolates the contribution of RA-CAD under a controlled and reproducible evaluation.

We acknowledge that a direct comparison with iterCAD (Hu et al., 2026) and ToolCAD (Gong et al., 2026) would strengthen the empirical validation, and we leave it for future work once their code, checkpoints, or evaluation pipelines become available. We have added this limitation explicitly and avoid claiming universal superiority over methods that could not be reproduced under the same setting.

H LIMITATIONS AND FUTURE WORK

Despite the promising results, RA-CAD has several limitations. First, it currently takes only a natural-language description as input, and direct visual conditioning on images, sketches, or point clouds is not supported, which limits its applicability to reverse engineering and visually grounded editing tasks. Second, the code space is restricted to the sketch-and-extrude DSL used by the training datasets, i.e., line, arc, and circle primitives with add/cut/intersect operations and 6-bit quantized parameters. Uncommon industrial operations, as well as continuous-parameter precision, are not covered, so generalization to broader CAD code spaces remains an open problem.

In future work, we plan to extend RA-CAD to multimodal inputs and interactive user feedback, enabling the agent to condition on images, sketches, or point clouds and to incorporate user edits during the design session. We will broaden the supported CAD code space to more diverse operations and finer quantization, and adapt the framework to additional execution environments and datasets.