

TOOLCAD: Exploring Tool-Using Large Language Models in Text-to-CAD Generation with Reinforcement Learning

Yifei Gong¹, Xing Wu^{1*}, Wenda Liu¹, Kang Tu¹,
¹School of Computer Engineering & Science, Shanghai University

Correspondence: xingwu@shu.edu.cn

Abstract

Computer-Aided Design (CAD) is an expert-level task that relies on long-horizon reasoning and coherent modeling actions. Large Language Models (LLMs) have shown remarkable advancements in enabling language agents to tackle real-world tasks. Notably, there has been no investigation into how tool-using LLMs optimally interact with CAD engines, hindering the emergence of LLM-based agentic text-to-CAD modeling systems. We propose TOOLCAD, a novel agentic CAD framework deploying LLMs as tool-using agents for text-to-CAD generation. Furthermore, we introduce an interactive CAD modeling gym to rollout reasoning and tool-augmented interaction trajectories with the CAD engine, incorporating hybrid feedback and human supervision. Meanwhile, an end-to-end post-training strategy is presented to enable LLMs to elicit refined CAD Modeling Chain of Thought (CAD-CoT) and evolve into proficient CAD tool-using agents via curriculum online reinforcement learning. Our findings demonstrate TOOLCAD fills the gap in adopting and training open-source LLMs for CAD tool-using agents, enabling them to perform comparably to proprietary models, paving the way for more accessible and robust autonomous text-to-CAD modeling systems.¹

1 Introduction

Computer-Aided Design (CAD) serves as a fundamental tool across the entire product lifecycle in modern industrial manufacturing, supporting continuous tracking and iterative refinement (Cherng et al., 1998). However, prototyping for industrial production from scratch still requires expert designers to manually perform accurate modeling with geometry-aware understanding, which is time-consuming and hinders the development of automation in complex CAD modeling. To enhance mod-

¹We make our project available on <https://gongyifeiisme.github.io/toolcad-project>.

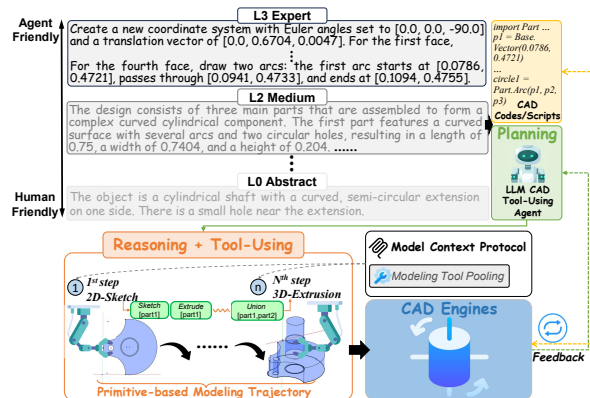


Figure 1: **Prompt-to-Tool vs. Prompt-to-Code.** L3 expert-level modeling text enables CAD tool-using agents to plan, reason, call tools, and complete modeling tasks via iterative CAD-engine feedback.

eling efficiency and enable automation, modern CAD products (e.g., *FreeCAD*, *SolidWorks*) provide APIs with procedural scripting codes such as Python for rapid modeling (Badagabettu et al., 2024). Besides, recent research has demonstrated that AI Agent for CAD generation, empowered by Large Language Models (LLMs) and Vision Language Models (VLMs), can increase designer productivity (Makatura et al., 2023; Kodnongbua et al., 2023). Nevertheless, despite these advances, there remains a significant gap between existing methods and a fully autonomous expert CAD modeling system that seamlessly integrates with design platforms (Zhou and Camba, 2025). Also, it remains unclear how to enable LLMs to directly interact optimally with CAD engines, rather than relying on token-based predictive modeling or generating CAD codes (Khan et al., 2024; Kolodiazhyi et al., 2026). Inspired by complex tool invocation in systems like Search-R1 (Jin et al., 2025) via LLM’s external tool-use capabilities, leveraging long-horizon reasoning with tool-integrated learning, we aim to develop LLM-based CAD tool-using agents capable of automatic modeling guided by

natural language commands (see Fig.1).

In contrast to agentic CAD generation methods (Mallis et al., 2024; Zhang et al., 2025; Wang et al., 2025c), our research explores the potential of LLMs leveraging reasoning and primitive-based tool usage for autonomous text-to-CAD generation. In this paper, the LLM acts as an expert tool-augmented CAD agent, tailored for decision-making in modeling steps and executing corresponding modeling tools (Xi et al., 2025; Qian et al., 2025). However, applying language agents to interact with the CAD engine and realize the full automation of CAD modeling workflow presents three key challenges: (1) Limited reasoning and tool-integration capabilities. (2) Lack of interactive CAD environments with modeling feedback, and agentic CAD tool-using evaluation benchmarks. (3) Insufficient training for proficient CAD tool-using agents across complex text-to-CAD tasks.

To address the aforementioned challenges, we propose TOOLCAD, an agentic CAD framework, by leveraging the in-context Chain-of-Thought prompting techniques and further online reinforcement learning (RL) to unlock the LLMs’ capabilities in step-level CAD Chain-of-Thought (CAD-CoT) reasoning and tool integration. This framework introduces an interactive CAD-specific modeling gym with hybrid feedback and human supervision, providing step-level and trajectory-level reward to enable LLMs to generate correct modeling tool-using trajectories through optimal interaction with the CAD engine. To further tackle CAD modeling tasks of varying complexity, we adopt a part-wise CAD curriculum exploration strategy and online GRPO optimization to improve the modeling stability of prompt-based agent policy, thereby developing robust and generalizable tool-using agents for complex text-to-CAD tasks.

Our contributions can be summarized as follows:

- We propose TOOLCAD, a novel framework that achieves full automation of text-to-CAD generation by leveraging tool-using LLM agents.
- TOOLCAD advances RL for tool-using LLM end-to-end training by introducing interactive CAD modeling gym with hybrid supervision from rule-based and outcome-based feedback signals. It provides a curriculum-based online exploration tool-learning strategy to develop more competitive CAD tool-using agents.
- We demonstrate that TOOLCAD outperforms

agentic generation baselines, enabling high-quality text-to-CAD results on held-out tasks across diverse complexities.

2 Related Work

Intelligent CAD Modeling System. Under the current trend of LLMs and VLMs demonstrating strong capabilities in complex reasoning and planning for real-world tasks, their integration into generative CAD modeling holds great potential for realizing intelligent and autonomous design systems. Recent advances on LLM-based CAD modeling approaches can be broadly categorized into two directions: (1) **LLM-based Parametric CAD Sequences Generation:** leveraging LLMs as auxiliary modules to assist token-level prediction for the next modeling command. Text2CAD (Khan et al., 2024) generates parametric CAD sequences from natural language instructions using a transformer-based network pretrained on multi-level CAD prompts annotated by Mistral and LLaVA-NeXT. CAD-GPT (Wang et al., 2025c) leverages the Multimodal Large Language Model (MLLM) LLaVA-1.5-7B, enhanced with 3D spatial reasoning capabilities, to precisely synthesize CAD modeling sequences. CAD-MLLM (Xu et al., 2024), the first intelligent CAD system to use a LLM Vicuna-7B to align multimodal input with modeling commands sequences for generating CAD models. (2) **LLM-based CAD Code Generation:** utilizing LLMs to generate and refine code-based modeling instructions for CAD reconstruction. CAD-Assistant (Mallis et al., 2024) uses GPT-4o with tool-augmented and docstring prompting to plan and generate code-level actions for autoconstraining and sketch parameterization. CAD-Llama (Li et al., 2025) augments CAD code generation using instruction-tuned LLaMA3-8B with Structured Parametric CAD Code (SPCC). CADCodeVerify (Alrashedy et al., 2025) prompts GPT-4o to make decisions on CAD design adjustments through an interactive question-answer feedback for code refinement. Seek-CAD (Li et al., 2026) integrates both visual and CoT feedback from DeepSeek-R1 and Gemini-2.0 to enable self-refinement of CAD code. Unlike these methods, TOOLCAD explores the optimal strategy for primitive-based tool-using LLM agents to interact with CAD engines for text-to-CAD generation.

Agentic Reinforcement Learning. Agent RL has shown promising results in training agents by aug-

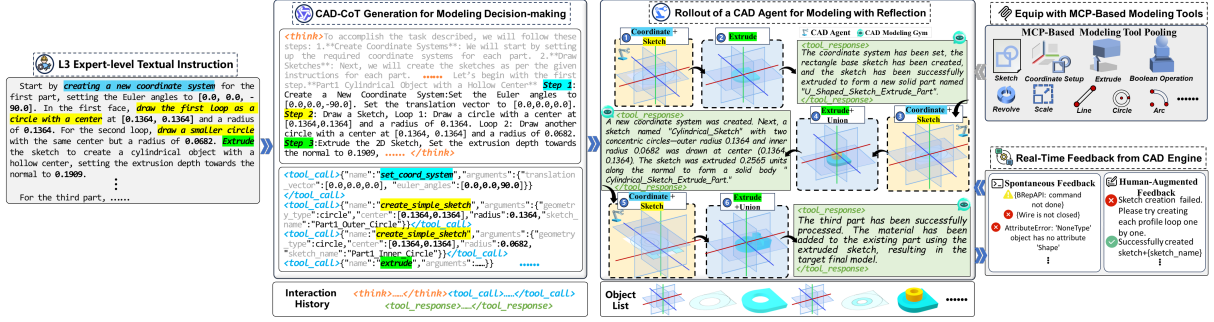


Figure 2: **CAD-specific Tool-Using Agent Workflow for Text-to-CAD Generation.** Given an expert-level natural language-based CAD design intent, the TOOLCAD framework performs 1) modeling decision-making, 2) equip with modeling tools and environment, 3) automatic modeling with reflection.

menting LLMs with the ability to invoke external tools for complex tasks (Wang et al., 2025d), advancing LLMs’ tool-integrated reasoning capabilities in interacting with external environments, such as retrieval engines and code interpret (Jin et al., 2025; Zhou et al., 2025). Early efforts on agent training explored classical RL algorithms such as DQN (Mnih et al., 2015), and later transitioned to value-based methods like PPO (Wang et al., 2025a) and AWR (Peng et al., 2019) for more stable optimization. Very recent approaches incorporate inference-time search strategy, such as Monte Carlo Tree Search (MCTS) (Yuan et al., 2025). Besides, training-based approaches employ Direct Preference Optimization (DPO) (Rafailov et al., 2023), Group Relative Policy Optimization (GRPO) (Singh et al., 2025; Shao et al., 2024) to align LLM-based agent rollout trajectory with human preference.

3 ToolCAD Workflow

3.1 Problem Formulation

Our goal is to develop an CAD-specific framework for adopting and training tool-using LLM to act as expert-level executors, tailored for various complex text-to-CAD tasks. From the view of language agent task, we model the process of tool-using agents’ autonomous thinking while performing tool-based CAD operations as a finite-horizon Markov Decision Process (MDP) $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}\}$, the state s denotes the historical context, which consists of reasoning along with the history of previous modeling actions. Given an expert-level instruction I , the agent policy π_θ must select a tool-based action a_t at any decision step t , guided on the current state s_t , then decide on the next action to transition to a new state s_{t+1} in a finite-horizon setting.

By introducing available real-time feedback from the CAD modeling gym, the tool-using trajectory along with reward path can be easily collected for training. Formally, the reasoning and tool integration of CAD modeling process is defined as:

$$a_t \sim \pi_\theta(\cdot | s_t, \tau_{<t}, I), \quad (r_t, s_{t+1}) \sim \mathcal{P}(\cdot | s_t, a_t) \quad (1)$$

where $\tau_{<t} = \{s_0, a_0, r_0, \dots, s_{t-1}, a_{t-1}, r_{t-1}\}$ denotes history interactions including all preceding reasoning-guided tool-calls, observations and rewards.

3.2 Framework Overview

The diagram in Fig.2 outlines the workflow of the tool-using LLM agent autonomously executing text-to-CAD tasks within the TOOLCAD framework. This framework incorporates three stages: **1) CAD Modeling Decision-making** – Utilizing chain-of-thought prompting and post-training, enable and evolve the capability of tool-using LLM to plan and decompose complex modeling tasks using CAD modeling Chain of Thought (CAD-CoT) generated in response to L3 expert-level prompts. **2) Equip with Modeling Tools and Environment** – The agent leverages the CAD-CoT, containing reasoning-guided actions, to call the corresponding custom-designed Model Context Protocol (MCP)-based modeling tools through interactions with the cad environment to advance the process of reasoning and tool integration. **3) Reflective Automatic CAD Modeling** – At this stage, we employ CAD-specific ReAct to maintain consistency across reasoning-guided actions and the CAD engine. Based on hybrid modeling feedback from the engine, the agent reflects on the outcome of each tool-call, adjusting and executing modeling tools iteratively until the task either succeeds or fails. The

detailed implementation of the tool-using agent can be found in the Appendix A.1, A.2, A.3.

3.3 CAD-CoT Reasoning and Tool Integration

CAD-CoT Prompting. Expert-level (L3) parametric CAD modeling instructions involve dense numerical parameters (e.g., coordinates, angles, radii, directions...), as well as a standardized and sequential procedural pipeline—spanning Coordinate Setup, Sketching, Extrusion and Boolean Operations (cut, union, common), to create each unit part. Constructing multi-part CAD models challenges tool-using LLMs’ long-horizon reasoning and tool integration, which significantly increases the risk of hallucinations in both parameters and action sequences. Hence, in order to elicit the tool-using agents to generate reliable CAD-CoT across multiple parts for complex text-to-CAD generation, we customize the CAD-specific ReAct (Yao et al., 2023) prompting strategy to ensure coherence between reasoning and tools. In addition, we instruct the LLM to structure CAD-CoT in a strict reasoning-guided modeling tool-call format, using special tokens (e.g., `<think>...</think>`, `<tool_call>...</tool_call>`), to reason out precise and reliable CAD modeling tool chains.

3.4 CAD Modeling Gym

This section presents the RL training components for the CAD tool-using agent, including the real-time hybrid modeling feedback and the trajectory collection pipeline.

CAD Modeling Feedback Design. To align text-based outputs with actual CAD execution, we first integrate an interactive and agentic modeling engines with LLMs via the open-source CAD platform FreeCAD by wrapping parametric CAD modeling primitives as agent-callable MCP-based modeling tools. The CAD compiler constitutes the core of the environmental feedback within our gym. Following the ReAct interaction paradigm, we design a step-level interactive modeling feedback mechanism from two key perspectives: **(1) Spontaneous Feedback.** In order to enable the agent receives coarse-grained feedback, we expose the CAD engine’s primitive-level tool responses including geometric conflict alerts, constraint warnings, and API error messages. This feedback manifests exceptions or error logs, facilitating the CAD agent to backtrack and self-correct along the modeling trajectory. **(2) Human-Augmented Feedback.** Because CAD engine’s spontaneous feedback mixes

code snippets and structured messages, it poorly reflects step-level modeling results, causing potential reasoning and tool-call hallucinations for tool-using agents. Therefore, we wrap the modeling tool outputs with human-augmented feedback, producing structured messages labeled as ‘*success*’ or ‘*fail*’. After each tool invocation, LLM-readable text descriptions of modeling results are stored in the interaction history s_t , enabling the agent to perform reflective CAD modeling based on observations \mathcal{O}_t . Specifically, the agent uses a geometric object list of actual CAD entities representing the current geometric state of the model to alleviate tool execution hallucinations, rather than relying solely on the recorded interaction history, ensuring reliable reflective modeling.

Demonstration Data Collection Pipeline. As illustrated in Fig.3, we construct a demonstration data collection pipeline to produce successful CAD tool-using trajectories from real-time interactions across tasks of varying complexity. Furthermore, to ensure alignment with target geometry, outcome-based evaluation is annotated through human visual check, with correction instructions introduced to guide the agent rollouts toward successful CAD modeling tool-using trajectories. This pipeline supports outcome-supervised reward model (ORM) training, enabling automated trajectory-level optimization in RL phases.

4 Post-training for Evolving CAD Agents

4.1 Reward Modeling.

As mentioned in Section 3.4, the CAD modeling gym provides a hybrid reward modeling mechanism that combines coarse-grained step-level feedback with outcome-supervised signals from the reward model \mathcal{M}_{ORM} .

ORM Training. Even with fine-grained engine feedback, the agent struggles to judge text-to-CAD generation quality and task completion based on the modeling tool-using trajectory. To address this, we fine-tune a LLM as the outcome-supervised reward model \mathcal{M}_{ORM} using demonstration data (including negative samples) annotated by human CAD modeling preference, to achieve task success evaluation and provide trajectory-level rewards. Subsequently, we wrap the instruction I and full modeling tool-using trajectory τ into the prompt to configure \mathcal{M}_{ORM} to response "YES" or "NO" to indicate whether a modeling tool-using trajectory successfully completes a target CAD geometry de-

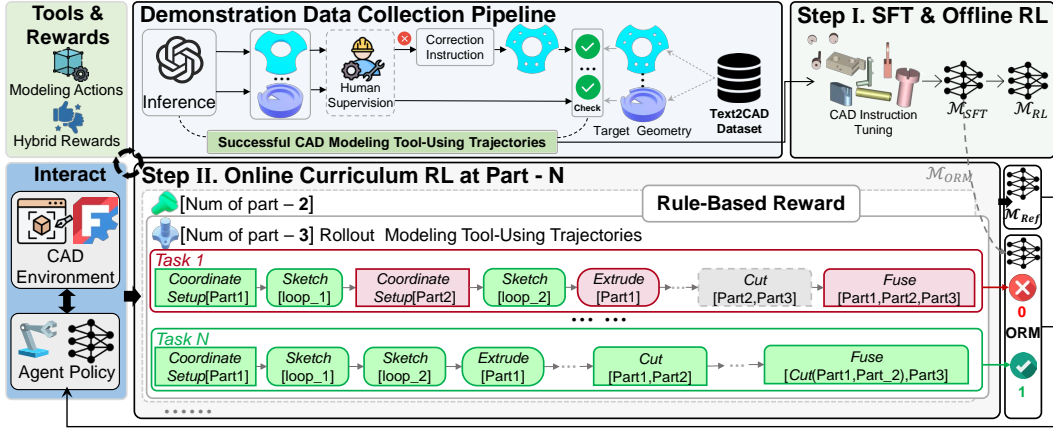


Figure 3: **Online-RL framework of TOOLCAD.** Starting from human-supervised CAD trajectories, the policy is refined into a robust agent through online curriculum reinforcement learning. The ORM training is consistent with human-supervised knowledge.

scribed by instruction I , leveraging the learned human knowledge from the language head of \mathcal{M}_{ORM} .

At the online reinforcement learning stage, \mathcal{M}_{ORM} serves as an automated metrics to access whether the agent’s rollout trajectory accomplishes a given task instruction, providing a binary reward signal (0 for failure and 1 for success). The reward is 1 if \mathcal{M}_{ORM} assigns a higher probability to “YES” than “NO”, and 0 otherwise.

Rule-Based Reward. Beyond encouraging convergence of generated tool-using trajectories to the ground-truth CAD reconstruction supervised by the final outcome reward, we incorporate step-wise rewards by extracting binary signals from feedback after each modeling tool invocation, which contains result labels (“success” or “fail”), providing coarse-grained reward signals through reward function \mathcal{R} :

$$\mathcal{R}(s_t, \mathcal{O}_t) = \text{EM}(\mathcal{O}_t, \text{label}) \quad (2)$$

Moreover, we introduce rule-based format rewards to enforce the tool-using agent policy structure its reasoning, tool use, and CAD environment interactions coherently and reliably, ensuring adherence to the prescribed CAD-CoT prompt template. The format reward function checks the correct order of reasoning and tool-using integration including reasoning (`<think>`), tool call (`<tool_call>`), and tool output (`<tool_response>`).

4.2 Training Strategy.

As depicted in Fig.3, we implement a two-stage agentic learning framework tailored for post-training CAD agents, consisting of supervised fine-tuning (SFT) followed by online curriculum reinforcement learning (RL), to evolve strong and

robust CAD tool-using agents.

Part-Wise CAD Curriculum Strategy. Because the unit number in a CAD model critically impacts modeling complexity, a potential challenge in training tool-using LLMs is the instability due to text-to-CAD generation with varying unit counts (Du et al., 2024). Overlength agent trajectories may lead to context overflow and catastrophic forgetting during rollouts. To stabilize online RL training, we design a part-wise CAD curriculum tool learning strategy that leverages action sequence average perplexity, gradually increasing task complexity by controlling the unit number in each CAD model. For a full trajectory τ , we use the actor π_θ to compute its perplexity:

$$\mathcal{P}(\tau) = \exp\left(-\frac{1}{T} \sum_{t=1}^T \log \pi_\theta(a_t | s_t)\right) \quad (3)$$

The optimization advances to the next curriculum learning stage once the average perplexity of the held-in test set drops below threshold δ , indicating sufficient policy confidence and proficiency on the current task complexity. Specifically, the perplexity threshold δ is softly set as a fraction α of the initial threshold, $\delta = \alpha \cdot \delta'$, where coefficient $\alpha \in (0, 1)$ controls over the exploration-exploitation trade-off in online RL.

Online RL-Evolving via CAD Exploration. We first utilize supervised fine-tuning to initialize the base agent policy model, resulting in \mathcal{M}_{SFT} , using successful CAD modeling tool-using trajectories from static demonstration data. We adopt online curriculum reinforcement learning across held-out tasks of varying complexity to enable the CAD tool-using agent to self-evolve, addressing imitation

learning’s lack of out-of-distribution generalization capability. Specifically, for each rollout modeling task trajectory $\tau_i = \{\tau_{i,(1)}, \dots, \tau_{i,(K)}\}$ of totally $|\tau_i|$ turns and K tokens, trajectory-level optimization enables critic-free training with GRPO strategy to update and its objective is:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \frac{1}{G} \sum_{i=1}^G \frac{1}{|\tau_i|} \sum_{k=1}^{|\tau_i|} \min \left[\frac{\pi_{\theta}(\tau_{i,k}|\tau_{i,<k})}{\pi_{\text{old}}(\tau_{i,k}|\tau_{i,<k})} \cdot \hat{A}_{i,k}, \text{clip} \left(\frac{\pi_{\theta}(\tau_{i,k}|\tau_{i,<k})}{\pi_{\text{old}}(\tau_{i,k}|\tau_{i,<k})}, 1 - \varepsilon, 1 + \varepsilon \right) \cdot \hat{A}_{i,k} - \beta \mathbb{D}_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}] \right] \quad (4)$$

where ε and β are hyperparameters, and $\hat{A}_{i,k}$ represent the relative advantage of the i -th task trajectory. The clipping threshold ε ensures stable updates. See more details in Appendix.B, and the training process in Appendix.C.3.

5 Experiment

5.1 Dataset and Evaluation Environment.

The effectiveness of prompting and learning framework is evaluated using the TOOLCAD environment along with dataset from the DeepCAD and Text2CAD. Text2CAD provides multi-level annotations from $\sim 170\text{K}$ models in the DeepCAD dataset, including four design prompts ranging from abstract to expert levels (L0~L3). We select and preprocess the L3 expert-level annotations as main text-to-CAD task instructions. In contrast to simple L0-L2, such long and precise L3 instructions are more consistent with industrial-level CAD output standards and agent-friendly. A major limitation of DeepCAD is the scarcity of complex tasks across multiple parts, with most of its $\sim 67\text{K}$ models containing only one part. Therefore, we construct tailored 982 offline demonstration trajectories via GPT-4o as held-in tasks with different levels of complexity for SFT initialization and ORM training. The rest of the dataset is reserved as held-out tasks for online RL training, and 200 test cases from the held-in task serve as the overall evaluation benchmark. More dataset and replication details are in the Appendix.C.1, C.2.

5.2 Baselines

Since there are no existing methods for training specific CAD tool-using agents to perform text-to-CAD modeling, we compare TOOLCAD with frontier proprietary LLMs utilizing prompting techniques, as well as open-sourced LLMs trained with alternative methods. For frontier LLMs, we select GPT-4o and Qwen3-235B-A22B, represent-

Model	Method	IR↓	MCD↓	Avg.SR(%)↑
<i>Frontier LLMs</i>				
GPT-4o	Zero-shot-CoT	20.49	19.84	48.4
	ReAct	9.12	5.88	<u>62.7</u>
Qwen3-235B-A22B	Zero-shot-CoT	30.25	37.18	51.3
	ReAct	25.65	29.83	60.5
<i>Open-Source LLMs</i>				
Qwen2.5-72B-Instruct	Zero-shot-CoT	52.94	50.73	36.1
	ReAct	45.17	48.29	43.4
Qwen3-8B	Zero-shot-CoT	60.39	61.78	18.5
	ReAct	54.02	53.18	27.9
	+SFT	16.13	29.43	41.2
	+AWR	24.58	33.64	37.8
	+TOOLCAD(ours)	<u>1.84</u>	<u>1.26</u>	61.8
	Zero-shot-CoT	70.32	68.41	14.9
Qwen-2.5-7B-Instruct	ReAct	62.88	61.39	23.4
	+SFT	15.21	28.49	43.7
	+AWR	22.74	30.77	39.2
	+TOOLCAD(ours)	1.51	1.12	63.9
<i>Transformer-based Generation Models</i>				
DeepCAD	–	15.36	13.41	–
TextCAD	–	2.25	1.97	–
SkexGen	–	4.59	3.85	–
HNC-CAD	–	3.45	3.08	–

Table 1: Main comparisons of average CAD modeling success rate(SR) and quality across different baselines. The **best** and second-best models are highlighted.

ing the current state-of-the-art in reasoning and agentic tool-use capabilities. For standard open-source models such as Qwen2.5-7B and Qwen3-8B, in addition to leveraging prompt-based reasoning and modeling-tool integration as baselines, we also train them using SFT and off-policy RL—advantage-weighted regression (AWR)(Peng et al., 2019), serving as RL-based evolution baselines for TOOLCAD. Additionally, Text2CAD, SkexGen(Xu et al., 2022) and HNC-CAD(Xu et al., 2023) represent mainstream approaches that generates CAD sequences, providing baselines for evaluating CAD generation quality. We further broadly compare TOOLCAD with prior agentic CAD generation methods (e.g., CAD-Assistant(Mallis et al., 2024), CAD-Llama(Li et al., 2025)) to highlight its agent-centric design and performance gains. Evaluation metrics are listed in the Appendix.C.4

5.3 Main Results

Overall Comparisons. Our main quantitative comparison results, presented in Table 1 and Fig.4, show that Qwen2.5-7B-Instruct and Qwen3-8B trained using TOOLCAD achieve average modeling success ratio of 63.9%, 61.8% respectively on CAD models with varying unit number, surpassing all SOTA frontier LLM prompting baselines. Meanwhile, GPT-4o (+14.3%) and Qwen3-235B-A22 (+9.2%) benefit significantly via TOOLCAD’s

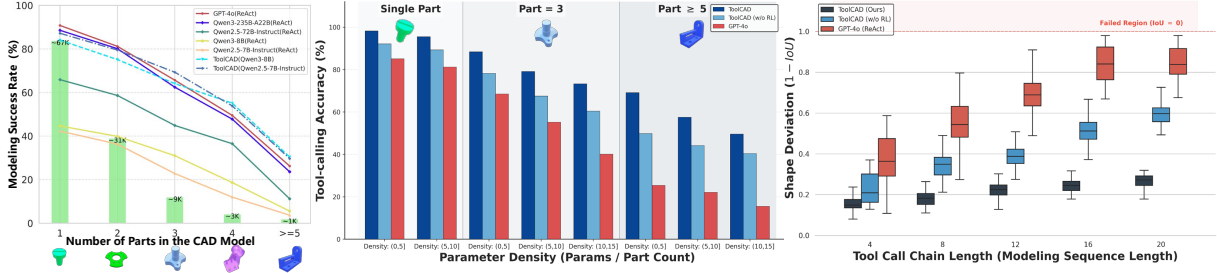


Figure 4: Evaluation of ToolCAD with RL. Modeling success rate, tool-calling accuracy, and geometric precision ($1 - IoU$) are evaluated to comprehensively measure the tool-using agent’s modeling behavior and performance.

Method	Fine-Tuning	VLM-Based	Backbone	Prompt Type	CAD-Code	ACC _T ↑	COV ↑	MMD ↓	JSD ↓
CAD-Assistant(Mallis et al., 2024)	×	✓(GPT-4o)	GPT-4o	L0 text + docstr + image	✓	51.25	60.32	6.13	25.09
CAD-GPT(Wang et al., 2025c)	✓	✓(LLaVA)	LLaVA	L0 text + image	×	52.78	64.59	4.49	22.17
CAD-Llama(Li et al., 2025)	✓	✓(CLIP)	LLaMA3-8B-Instruct	L0~L1 text	×	79.46	74.86	1.62	3.85
VideoCAD(Man et al., 2025)	✓	✓(DINOv2/CLIP)	ViT	video frame	×	—	49.25	11.75	32.19
FlexCAD(Zhang et al., 2025)	✓	×	LLaMA3-8B-Instruct	L3 text token	×	81.43	70.49	2.35	4.79
CADCodeVerify(Alrashedy et al., 2025)	×	✓(GPT-4/Gemini-1.5-Pro)	GPT-4/Gemini-1.5-Pro/CodeLlama	L2~L3 text	×	—	64.37	5.46	18.26
Text2CAD(Khan et al., 2024)	×	×	BERT	L0~L3 text	×	60.39	61.94	5.19	10.28
CAD Translator(Li et al., 2024)	✓	×	Transformer	L0 text	×	69.28	65.22	3.79	9.14
CADFusion(Wang et al., 2025b)	✓(+DPO)	✓(GPT-4o)	LLaMA3-8B-Instruct	L1~L2 text	×	72.65	71.93	3.07	5.03
CAD-Coder(Guan et al., 2025)	✓(+GRPO)	×	Qwen2.5-7B-Instruct	L0~L3 text	✓	57.34	65.48	5.73	11.42
RLCAD(Yin et al., 2025)	✓(+PPO)	×	Transformer	/	×	—	55.18	7.45	8.33
TOOLCAD(ours)	✓(+GRPO)	×	Qwen2.5-7B-Instruct	L3 text	×	80.63	79.06	1.36	3.21

Table 2: Comparison of agentic CAD generation methods using vision-language models (VLMs) and large language models (LLMs) across multi-part task with varying complexity.

Methods	Feedback Source	1P	3P	5+P
CAD-Assistant (GPT-4o)	Visual+Docstr	82.5	59.1	20.8
CAD-GPT (LLaVA)	Visual+Text	80.3	54.8	18.4
TOOLCAD-VLM (variant)	Visual+Text	85.6	50.3	15.9
TOOLCAD (Ours)	Text	87.2	69.3	29.7

Table 3: VLM-based feedback vs. Engine-based feedback success rate (%)

specific ReAct CAD-CoT prompting, achieving notably higher modeling success rates than with simple prompt-based baseline Zero-shot-CoT. This results highlight that TOOLCAD enables LLMs to act as effective tool-using agents for text-to-CAD tasks. For learning-based baselines, the two-stage post-training strategy of TOOLCAD outperforms other initial method, such as SFT and offline-RL AWR, remarkably after our offline-to-online RL evolution phase, achieving about 20% absolute improvement in average success rate on open-source LLMs, from 41.2% and 43.7% (SFT) to 61.8% and 63.9% (Online RL), demonstrating that online interactive exploration with the CAD engine results in more robust agent policy and mitigates the generalization risk in off-policy learning.

Performance with Varying Complexity. As shown in Fig.4, as the part number of CAD model increases, prompt-based tool-using LLMs struggle on multi-part tasks, including GPT-4o. In contrast, online-RL trained models show consistent improvements. For instance (figure left), the reasoning-

based model Qwen3-8B surpasses GPT-4o on three-part tasks and achieves the best performance among all baselines on 4+ part tasks, attributed to its long-horizon reasoning ability. TOOLCAD achieves the highest tool-calling accuracy, particularly in tasks with high parameter density (figure mid). This indicates that the TOOLCAD’s tool-using agent effectively learns robust instruction-following capabilities in complex modeling, whereas TOOLCAD (w/o RL) struggles significantly as parameter complexity scales. To assess progress toward high-quality modeling, we utilize CAD-engine-verifiable metric ($1 - IoU$, figure right). TOOLCAD achieves the minimum geometric error in long tool call chains, whereas GPT-4o suffers from spatial hallucination as sequences scale.

Broader Comparison in Agentic CAD Generation. Table 2 shows TOOLCAD outperforms major baselines on multi-part text-to-CAD tasks via L3 text prompt. Most baselines employ VLMs as either judges or feature encoders, but suffer from hallucination and domain mismatch: they appear effective on simple toys yet fail on harder out-of-distribution text-to-CAD tasks. VLM-based methods introduce not transparent feedback that severely undermines robustness. Table 3 demonstrates that VLM-based methods, including our variant ToolCAD-VLM, perform reasonably well on simple geometries but are more prone to accumulating spatial hallucinations during long-horizon

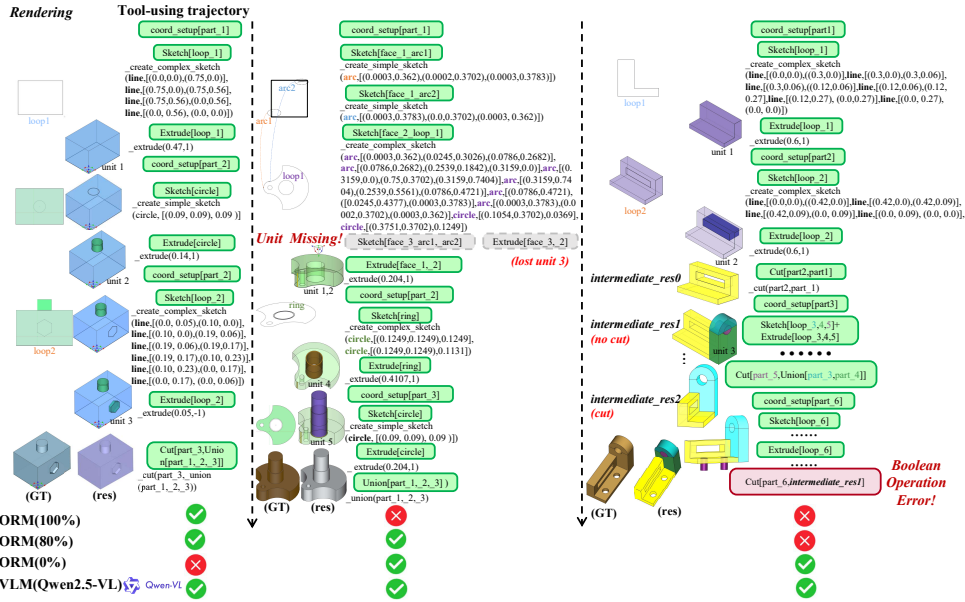


Figure 5: Case study of tool-using agent modeling trajectories. Our ORM (100%—trained) detects subtle step-level near-miss failures such as "Unit Missing" and "Boolean Error" that are visually indistinguishable to the VLM judge (Qwen2.5-VL).

Methods	Instruction-level	COV \uparrow	JSD \downarrow	Novelty \uparrow	IoU $_{best}\uparrow$
Text2CAD	@L0	40.27%	36.49	94.33%	40.51%
	@L1	49.35%	20.21	92.58%	45.39%
	@L2	54.68%	17.15	90.53%	54.77%
	@L3	61.94%	10.28	88.47%	61.24%
TOOLCAD (Ours)	@L0	58.19%	20.81	98.14%	46.63%
	@L1	61.17%	11.25	96.31%	53.91%
	@L2	70.29%	6.47	91.75%	60.35%
	@L3	79.06%	3.21	91.62%	65.83%

Table 4: Comparison on geometry quality and diversity across different instruction levels.

reasoning across CAD design tasks of varying complexity (*1P: Simple, 3P: Medium, 5+P: Hard*). TOOLCAD eliminates VLMs entirely, text-driven feedback and agent behavior that is straightforward and stable.

Instruction-level Geometric Evaluation. To

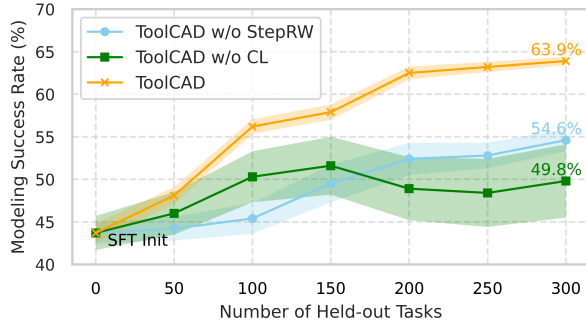


Figure 6: Ablation study of online RL framework on step-level feedback and part-wise curriculum learning strategy.

measure the geometric alignment between the target CAD model and the tool-using agent’s output, we evaluate the final CAD reconstructions in terms of geometric accuracy and overall quality. Table 1 reports the quantitative geometric modeling quality of TOOLCAD and baseline methods under L3 expert-level instructions. The post-training methods consistently outperform transformer-based generation baselines DeepCAD and Text2CAD. In order to validate TOOLCAD’s intrinsic geometric unfolding capability, we conduct experiments on generative quality and diversity under vague L0/L1 instructions (Table 4). The results demonstrate that TOOLCAD is not restricted to L3-level precision. Instead, it remains highly competitive under instruction noise by leveraging its learned modeling paradigm to autonomously plan trajectories that achieve the target geometry. Furthermore, we study the effect of CAD agent’s the instruction dependence on instruction-level geometric model-

ing quality using F1 score including sketch (*Line*, *Arc*, *Circle*) and extrusion, and the results are summarized in Table 5. Text2CAD is trained using multi-level instructions from L0~L3 to realize all skill level consistent modeling. In contrast, our framework generalizes to non-expert instructions via the internalized structure of task trajectories, which empowers the agent to unfold noisy L1 and L2 instructions from abstract intent without explicit tool-call cues.

5.4 Ablation Studies

Method	Instruction-level					
	@L1		@L2		@L3	
	Sketch	Extrusion	Sketch	Extrusion	Sketch	Extrusion
Text2CAD	30.35	57.19	47.25	69.31	58.47	88.62
TOOLCAD(Qwen2.5-7B)	24.59	44.22	43.23	75.26	78.91	93.44
TOOLCAD(Qwen3-8B)	27.58	49.25	45.69	77.86	81.52	95.21

Table 5: Instruction-level Sketch and Extrusion F1 score.

	Our ORM	GPT-4o	Qwen3-235B	Qwen2.5-VL
Test Cases (%)	82.7	75.9	70.4	55.3
Hard Cases (%)	65.2	54.7	50.8	41.6

Table 6: Performance of ORMs.

Evaluation of ORM. In our RL-training framework, ORM (based on Qwen2.5-7B) plays a crucial role in evaluating modeling trajectories to guide the agent’s learning process. Hence, we assess ORM’s effectiveness on the 200 test cases from successful demonstration trajectories and additional failed trajectories, with a particular focus on hard cases involving five or more units (≥ 5). We compare its performance with several baseline models, including proprietary LLM APIs GPT-4o, Qwen3-235B-A22B and Qwen2.5-VL. Table 6 indicates that our ORM substantially outperforms baselines and retains effective supervision on complex tasks.

Impact of Online RL Optimization Components. To assess the contribution of key components in our online RL framework, we conduct ablation studies on the step-wise reward mechanism and the part-wise curriculum learning strategy. Accordingly, we implement two variants: TOOLCAD w/o StepRW, which removes the step-level rule-based reward function, and TOOLCAD w/o CL, which disables the part-wise curriculum learning schedule.

The results, shown in Fig.6, demonstrate that all the components are critical for stability and improvement in the agent’s online exploration.

(1) The effect of the step-wise reward. The results indicates that a substantial performance degradation when this rule-based reward function is removed. This highlights the sparse supervision provided by ORM alone is insufficient to support effective optimization of step-level modeling tool-using trajectories. **(2) The effect of the part-wise curriculum learning strategy.** Compared with TOOLCAD w/o CL, TOOLCAD shows faster progression and consistently higher performance as the number of hold-out tasks increases. Without CL, the agent lacks structured difficulty guidance and struggles when task complexity varies randomly, especially beyond its current skill horizon. Additional analyses are provided in Appendix.D

6 Case Study

Fig.5 shows three cases in long-horizon multi-part CAD task. Case 1 shows a correct tool call chain for three-part reconstruction, where ORM (100%, 80%) and VLM validator accurately judges success. Cases 2–3 present modeling near-miss failures whose geometry is visually close to ground truth (GT), but the VLM fails to distinguish the subtle visual discrepancies between generated results and GT in the rendered views. Case 2 contains a missing unit failure, detected only by ORM (100%), demonstrating its strength in step-level trajectory supervision. Case 3 illustrates a Boolean Operation error under long tool chaining, highlighting its strong dependence on the tool-using agent’s long-horizon reasoning and planning. The VLM misjudges due to negligible visual cues and subtle blind spots during the CAD design process. More visual results are provided in Appendix.D.4.

7 Conclusion

In this paper, we propose TOOLCAD, a novel LLM-based tool-using agent intelligent CAD system that automates the text-to-CAD modeling framework. This features an interactive CAD gym for agentic reasoning and tool-based interaction, coupled with a curriculum reinforcement learning pipeline that enables LLMs to evolve into proficient CAD tool-using agents. Experimental results suggest that TOOLCAD enables open-source LLMs to generalize across complex modeling tasks, supporting their potential as effective backbones for autonomous CAD systems.

Limitations

Although empirical experiments have confirmed the effectiveness of the proposed TOOLCAD, two major limitations remain. First, while we enable efficient L3 text-prompt-driven tool-using agent workflows for text-to-CAD generation, incorporating visual cue-guided perception as an upstream step to simplify tool calling would be highly valuable. Second, our exploration of CAD-specific tool learning is still limited—lack of self-correction, ORM performance degradation and a limited tool library can hinder task execution. Designing a more reliable Agent–CAD engine interaction framework with stronger geometric feedback may be a more promising future direction than tool learning alone.

Ethical Statement

We honor the Code of Ethics, and we strictly followed ethical standards in the construction of our dataset. No private data or non-public information is used in our work.

References

- Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. 2025. [Generating cad code with vision-language models for 3d designs](#). In *The Thirteenth International Conference on Learning Representations*.
- Akshay Badagabettu, Sai Sravan Yarlagadda, and Amir Barati Farimani. 2024. [Query2cad: Generating cad models using natural language queries](#). *Preprint, arXiv:2406.00144*.
- John G Cherng, Xin-Yu Shao, Yubao Chen, and Peter R Sferro. 1998. [Feature-based part modeling and process planning for rapid response manufacturing](#). *Computers & industrial engineering*, 34(2):515–530.
- Anna C Doris, Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. 2026. [Cad-coder: An open-source vision-language model for computer-aided design code generation](#). *Journal of Mechanical Design*, 148(7):071702.
- Yuhao Du, Shunian Chen, Wenbo Zan, Peizhao Li, Mingxuan Wang, Dingjie Song, Bo Li, Yan Hu, and Benyou Wang. 2024. [Blenderllm: Training large language models for computer-aided design with self-improvement](#). *Preprint, arXiv:2412.14203*.
- Yandong Guan, Xilin Wang, Xingxi Ming, Jing Zhang, Dong Xu, and Qian Yu. 2025. [Cad-coder: Text-to-cad generation with chain-of-thought and geometric reward](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. [Search-r1: Training llms to reason and leverage search engines with reinforcement learning](#). *Preprint, arXiv:2503.09516*.
- Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. 2024. [Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, volume 37, pages 7552–7579.
- Milin Kodnongbua, Benjamin T Jones, Maaz Bin Safeer Ahmad, Vladimir G Kim, and Adriana Schulz. 2023. [Zero-shot cad program re-parameterization for interactive manipulation](#). In *Proceedings of the ACM SIGGRAPH Asia 2023*. ACM.
- Maksim Kolodiaznyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. 2026. [cadrille: Multi-modal cad reconstruction with online reinforcement learning](#). *Preprint, arXiv:2412.14203*.
- Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. 2025. [Cad-llama: leveraging large language models for computer-aided design parametric 3d model generation](#). In *Proceedings of the Forty-second Computer Vision and Pattern Recognition Conference*, pages 18563–18573.
- Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. 2026. [Seek-cad: A self-refined generative modeling for 3d parametric cad using local inference via deepseek](#). In *The Fourteenth International Conference on Learning Representations*.
- Xueyang Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. 2024. [Cad translator: An effective drive for text to 3d parametric computer-aided design generative modeling](#). In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 8461–8470.
- Liane Makatura, Michael Foshey, Bohan Wang, Felix Hähnlein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, and 1 others. 2023. [How can large language models help humans in design and manufacturing?](#) *Preprint, arXiv:2307.14377*.
- Dimitrios Mallis, Ahmet Serdar Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. 2024. [Cad-assistant: Tool-augmented vllms as generic cad task solvers](#). *Preprint, arXiv:2412.13810*.
- Brandon Man, Ghadi Nehme, Md Ferdous Alam, and Faez Ahmed. 2025. [Videocad: A large-scale video dataset for learning ui interactions and 3d reasoning from cad software](#). *Preprint arXiv:2505.24838*.

- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fiedler, Georg Ostrovski, and 1 others. 2015. [Human-level control through deep reinforcement learning](#). *nature*, 518(7540):529–533.
- Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. 2019. [Advantage-weighted regression: Simple and scalable off-policy reinforcement learning](#). *Preprint, arXiv:1910.00177*.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiushi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. [Toolrl: Reward is all tool learning needs](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- 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 *The Thirty-seventh Annual Conference on Neural Information Processing Systems*, volume 36, pages 53728–53741.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *Preprint, arXiv:2402.03300*.
- Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. [Agentic reasoning and tool integration for llms via reinforcement learning](#). *Preprint, arXiv:2505.01441*.
- Hongru Wang, Cheng Qian, Wanjun Zhong, Xiushi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025a. [Acting less is reasoning more! teaching model to act efficiently](#). *Preprint, arXiv:2504.14870*.
- Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. 2025b. [Text-to-cad generation through infusing visual feedback in large language models](#). In *Forty-second International Conference on Machine Learning*.
- Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. 2025c. [Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms](#). In *Proceedings of the Thirty-ninth AAAI Conference on Artificial Intelligence*, volume 39, pages 7880–7888.
- Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, and 1 others. 2025d. [Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning](#). *Preprint, arXiv:2504.20073*.
- Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Dingwen Yang, Chenyang Liao, Xin Guo, Wei He, and 1 others. 2025. [Agentgym: Evolving large language model-based agents across diverse environments](#). In *Proceedings of the Sixty-third Annual Meeting of the Association for Computational Linguistics*, volume 1, page 27914–27961.
- Jingwei Xu, Zibo Zhao, Chenyu Wang, Wen Liu, Yi Ma, and Shenghua Gao. 2024. [Cad-mlm: Unifying multimodality-conditioned cad generation with mlm](#). *Preprint, arXiv:2411.04954*.
- Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. 2023. [Hierarchical neural coding for controllable cad model generation](#). In *Proceedings of the Fourteenth International Conference on Machine Learning*.
- Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chinyi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. 2022. [Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks](#). In *Proceedings of the Thirty-ninth International Conference on Machine Learning*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [React: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations*.
- Xiaolong Yin, Xingyu Lu, Jiahang Shen, Jingzhe Ni, Hailong Li, Ruofeng Tong, Min Tang, and Peng Du. 2025. [Rlcad: Reinforcement learning training gym for revolution involved cad command sequence generation](#). *Preprint, arXiv:2503.18549*.
- Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. [Agent-r: Training language model agents to reflect via iterative self-training](#). *Preprint, arXiv:2501.11425*.
- Zhanwei Zhang, Shizhao Sun, Wenxiao Wang, Deng Cai, and Jiang Bian. 2025. [Flexcad: Unified and versatile controllable cad generation with fine-tuned large language models](#). In *The Thirteenth International Conference on Learning Representations*.
- Jiwei Zhou and Jorge D Camba. 2025. [The status, evolution, and future challenges of multimodal large language models \(llms\) in parametric cad](#). *Expert Systems with Applications*, page 127520.
- Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. 2025. [Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks](#). *Preprint, arXiv:2503.15478*.

A CAD Agent Implementation Details

A.1 Prompt Templates

The complete prompt designs for the CAD Tool-Using Agent and the ORM model \mathcal{M}_{ORM} are presented below.

A.2 Integration of LLM Agents into FreeCAD via the Model Context Protocol

To efficiently equip the CAD agent with modeling tools and environment, as shown in the **Figure.11**, we implement an MCP (Model Context Protocol) Server for the FreeCAD platform to standardize the interfaces and invocation processes of modeling tools. The MCP client facilitates text-to-instruction automatic modeling interactions, enabling the CAD agent to scale and evolve its modeling capabilities.

For custom-designed modeling tools, we implement and encapsulate a set of CAD modeling tools based on the MCP-Server to construct a comprehensive tool library as follows:

$$\text{TOOL LIBRARY} = \left\{ \begin{array}{l} \text{freecad-set_coord_system,} \\ \text{freecad-create_complex_sketch,} \\ \text{freecad-create_simple_sketch,} \\ \text{freecad-boolean_operation,} \\ \text{freecad-multiple_fuse,} \\ \text{freecad-extrude_face} \end{array} \right\}$$

A.3 Feedback Interface Design

The detailed design examples of the Human-Augmented Feedback interface are shown in **Figure.12** and **Figure.13**, corresponding to the feedback involved in the primitive-based tool *create_complex_sketch* and *bool_operation*, respectively.

B Details of Policy Update Algorithm in Post-training

B.1 Online RL Optimization Details

We formulate the RL objective function utilizing a CAD engine (CAD environment) \mathcal{E}_{CAD} as follows:

$$\max_{\pi_{\theta}} \mathbb{E}_{\mathcal{M}, I \sim \mathcal{D}, \tau \sim \pi_{\theta}(\cdot | I, s; \mathcal{E}_{\text{CAD}})} [\mathcal{R}_{\phi}(\tau)] - \beta \mathbb{D}_{\text{KL}}[\pi_{\theta}(\tau | I, s; \mathcal{E}_{\text{CAD}}) || \pi_{\text{ref}}(\tau | I, s; \mathcal{E}_{\text{CAD}})] \quad (5)$$

\mathcal{R}_{ϕ} is the reward function and \mathbb{D}_{KL} is KL-divergence and β controls regularization. The overall training objective remains the maximization of the expected outcome-based reward $\mathcal{R}_{\phi}(\tau_i)$ from a reference policy π_{ref} to maintain stability. For

critic-free training leveraging GRPO, the normalized reward $\hat{A}_{i,k}$ is shared across all tokens in τ_i :

$$\hat{A}_{i,k} = \frac{\mathcal{R}(\tau_i) - \max(\{\mathcal{R}(\tau_1), \dots, \mathcal{R}(\tau_G)\})}{\text{std}(\{\mathcal{R}(\tau_1), \dots, \mathcal{R}(\tau_G)\})} \quad (6)$$

where G is the number of modeling trajectories in the batch, hence, $\hat{A}_{i,k}$ denotes the advantage at token k in τ_i .

B.2 Modeling Behavioral Cloning with Collected Trajectories

Specifically, we employ behavioral cloning, also referred to as supervised fine-tuning (SFT), to fine-tune LLM-based CAD tool-using agents by having them mimic the collected modeling trajectories. The training data is derived from demonstration data collection pipeline, sourced from the our proposed CAD modeling gym. We use these collected trajectories to train a base generally-capable CAD agent with basic instruction-following and CAD modeling abilities. We maximize the as the initial policy:

$$\mathcal{J}_{\text{BC}}(\theta) = \mathbb{E}_{\tau \sim \mathcal{D}} \sum_{t=1}^T [\log \pi_{\theta}(a_t | s_{t-1})]. \quad (7)$$

where \mathcal{D} denotes collected modeling trajectory data.

B.3 Reward Design

In TOOLCAD, the final optimization objective for GRPO relies on an aggregated scalar reward R , which is a weighted combination of trajectory-level and step-wise signals. Specifically, for each generated trajectory t_i in a group of size G , the total reward $R(t_i)$ is defined as:

$$R(t_i) = \alpha \cdot R_{\text{ORM}} + \beta \cdot \left(\frac{1}{T} \sum_{t=1}^T R_{\text{step}}^{(t)} \right) + \gamma \cdot R_{\text{format}} \quad (8)$$

- **Outcome Reward:** A trajectory-level terminal reward (0 or 1) assessed by the ORM.
- **Step-wise Execution Reward:** For each step t , we assign a binary reward $R_{\text{step}}^{(t)} \in \{0, 1\}$. A reward of 1 is granted only if the CAD engine returns a ‘‘Success’’ status for the primitive execution. We compute the mean over T steps to maintain a consistent reward scale regardless of trajectory length.

CAD Tool-Using Agent Prompt Template

You are an expert Computer Aided Design (CAD) modeling agent that can fulfill user's high-level instructions. Given modeling instructions, you plan the steps of the CAD modeling task and use provided modeling tools to complete the intermediate parts of a final CAD model. You are provided with modeling tool signatures within `<tools></tools>` XML tags:

`<tools>`

{modeling tools and descriptions}

`</tools>`

Each part is typically built by:

1. Creating a coordinate system.
2. Drawing a 2D sketch.
3. Extruding it into a 3D shape.
4. Optionally applying Boolean Operations.

You always first plan the steps of the CAD modeling process by wrapping your reasoning in `<think>` and `</think>`.

For each function call, return a json object with function name and arguments within `<tool_call></tool_call>` XML tags:

`<tool_call>`

"name": <function-name>, "arguments": <args-json-object>

`</tool_call>`

Once all parts are built and assembled into a final model named 'FinalModel', respond with: `<answer>COMPLETED</answer>`

#REMEMBER:

- No need to provide feedback on how to manually complete the modeling operations.
- Flexibly use Boolean Operations such as cut out, fuse, and common.

- **Format Reward:** Checks whether the model output contains all required special tokens in the correct order. Specifically, an additional reward of 0.5 is awarded if: (1) all tags are present, (2) the internal order of opening/closing tags is correct, and (3) the overall structure follows:

`<think>` → `<tool_call>` → `<tool_response>`.

C Demonstration Dataset Details

C.1 Collection Pipeline and Data Split

In our static demonstration trajectory collection pipeline, different frontier proprietary LLMs (GPT-

4o, Qwen3-235B-A22B, Gemini-1.5-pro, Qwen3-32B) are employed to perform agent inference for various text-to-CAD tasks, aiming to generate a diverse distribution of modeling tool-using trajectories. Each generated trajectory is visually checked and aligned with the corresponding ground-truth CAD model under expert supervision. When current geometry errors or omissions are identified, correction instructions are provided to prompt the tool-using agent in correcting CAD-CoT and completing missing steps, thereby ensuring the overall correctness of the trajectories. In total, 982 successful modeling-using trajectories with varying complexity are collected. To ensure a balanced

Prompts for \mathcal{M}_{ORM} to Assess the Task Trajectory

You are an expert in evaluating the performance of CAD modeling agent. You will be given a modeling historical context log that memorizes an CAD agent interacting with the CAD environment to automatic CAD modeling, and the designer’s intent, the format of history modeling interactions is:

###

Actions: Action trajectory

Obverations: Refletcion

###

The Designer Intent: {instruction}

Action History: {modeling histical interaction context}

Your goal is to judge whether the agent’s execution is successful or not. You must respond with **YES** or **NO**.

Number of Part	#Train Traj.	#Test Traj.
Part-1	100	40
Part-2	130	40
Part-3	300	40
Part-4	130	40
Part- ≥ 5	122	40
Total	782	200

Table 7: The statistics of held-in modeling tool-using trajectories.

evaluation across tasks of varying complexity, we perform stratified sampling when partitioning the 982 collected trajectories into training and test sets. Specifically, we divide the data into five complexity levels (Part-1 through Part- ≥ 5), and allocate a fixed number of 40 trajectories per level to the test set. The remaining trajectories are assigned to training, resulting in 782 training trajectories and 200 test trajectories. This stratification guarantees that the test set remains uniformly distributed across complexity tiers, while preserving sufficient samples per tier in the training set for effective learning. The detailed statistics are shown in Table 7. A subset of 200 trajectories is selected from the 982 collected demonstrations as test cases for held-in tasks, serving as evaluation benchmarks for SFT, online RL, and ORM models.

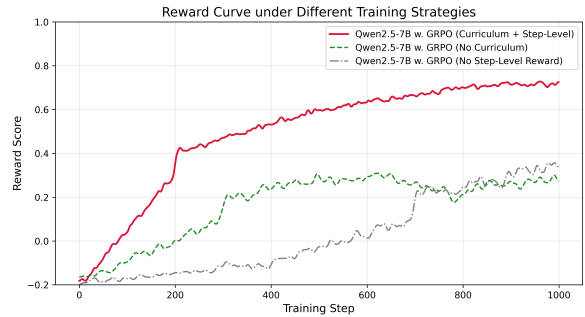


Figure 7: Comparison of different training strategies.

C.2 Implementation Details

All experiments are conducted using $8 \times \text{H20-96GB}$ GPUs. The detailed training process can be found in Algorithm 1. The learning rate for iterative SFT is $1e - 5$, with 0.1 warm-up ratio and a cosine scheduler. In online RL phase, we sample 8 rollouts per modeling task instruction with a temperature of 0.1. We set a high generation budget of 8,192 tokens to accommodate long-horizon, multi-step reasoning and tool-using. The maximum context window is set to 16384 tokens, and the maximum response length per rollout is 512 tokens. Training is performed using GRPO with a batch size of 32 and a learning rate of $2e - 6$. Our codebase is built on Verl and LLaMA-Factory.

C.3 Training Procedure

The overall pipeline of the proposed algorithm is illustrated in Algorithm.1.

C.4 Metrics

To thoroughly evaluate TOOLCAD’s generation quality and the modeling performance of the tool-using agent, we use these metrics:

- *Invalidity Ratio (IR)*: quantifies the percentage of the output CAD models that fail to be converted to point clouds.
- *Chamfer Distance (CD)*: calculates point-wise proximity between point clouds sampled from \mathcal{X} and \mathcal{Y} :

$$d_{CD}(\mathcal{X}, \mathcal{Y}) = \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \min_{y \in \mathcal{Y}} \|x - y\|_2^2 + \frac{1}{|\mathcal{Y}|} \sum_{y \in \mathcal{Y}} \min_{x \in \mathcal{X}} \|y - x\|_2^2 \quad (9)$$

- *Minimum Matching Distance (MMD)*: quantifies the average distance between the generated model and its closest-matching reference shape.
- *Intersection over Union (IoU)*: as the foundational metric for measuring global volumetric alignment between the generated model \mathcal{G} and the reference model \mathcal{S} .

$$\text{IoU}(\mathcal{G}, \mathcal{S}) = \frac{\mathcal{G} \cap \mathcal{S}}{\mathcal{G} \cup \mathcal{S}}, \quad (10)$$

- *Parameter Density*: This metric calculates the average complexity per shape, defined as:

$$PD = \frac{N_{param}}{N_{unit}}, \quad (11)$$

N_{param} denotes the total parameter count (e.g., coordinates, radii, and angles) and N_{unit} represents the total number of parts in a CAD model.

- *Coverage (COV)*: measures how well the generative model covers the real data distribution.

$$\text{COV}(\mathcal{G}, \mathcal{S}) = \frac{|\{\arg \min_{\mathcal{Y} \in \mathcal{S}} d_{CD}(\mathcal{X}, \mathcal{Y}) | \mathcal{X} \in \mathcal{G}\}|}{|\mathcal{S}|} \quad (12)$$

- *Jensen-Shannon Divergence (JSD)*: measures the dissimilarity between two point clouds from the perspective of voxel distribution.

$$\text{JSD}(P_{\mathcal{G}}, P_{\mathcal{S}}) = \frac{1}{2}D(P_{\mathcal{S}} \parallel M) + \frac{1}{2}D(P_{\mathcal{G}} \parallel M), \quad (13)$$

where $M = \frac{1}{2}(P_{\mathcal{S}} + P_{\mathcal{G}})$ and D is the KL-divergence. $P_{\mathcal{G}}$ and $P_{\mathcal{S}}$ are distributions of

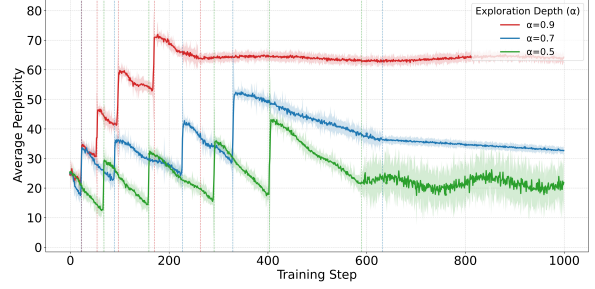


Figure 8: Average perplexity vs. threshold factor α during online exploration.

points in the generated and reference models, respectively.

- *IOU_{best}*: We adopt the IoU_{best} metric proposed in (Doris et al., 2026) for evaluating solid geometry similarity.

D More Quantitative Result

D.1 Training Process

Figure.7 compares three reinforcement-learning strategies. Curriculum + Step-Level Reward (red) achieves the fastest and most stable improvement, showing the benefit of combining dense step-level feedback with progressive task difficulty. No Curriculum (green), which retains ORM and step-level rewards but removes difficulty scheduling, converges more slowly and exhibits higher variance. In contrast, No Step-Level Reward (gray), which relies solely on a final binary ORM reward, performs poorly due to sparse feedback and unstable credit assignment. These results highlight that step-level supervision is critical for stable learning, while curriculum further enhances convergence.

D.2 Extra Ablations

The impact of perplexity. We investigate how the coefficient α influences average perplexity during online exploration on held-out modeling tasks, providing insights into its role in modulating exploration depth and the underlying exploration-exploitation trade-off for training proficient CAD agents, as shown in Fig. 8. The study is conducted on curriculum learning strategy with three threshold coefficients $\alpha = 0.9, 0.7, 0.5$. When $\alpha = 0.9$, the agent quickly reaches final-stage tasks (part count ≥ 5) due to low exploration depth, but maintains high perplexity and fails to improve, indicating insufficient skill acquisition. In contrast, with $\alpha = 0.5$, he agent explores more deeply but with

Model	Avg Tokens	Avg Latency (ms)
TOOLCAD(Qwen2.5-7B)	5178	648
TOOLCAD(Qwen3-7B)	6493	788
GPT-4o	11703	982
Qwen3-235B-A22B	9820	1108
CAD-Assisant(GPT-4o)	30802	1326

Table 8: Comparison of average token usage per tool-call and average tool-call latency across different models.

higher time consumption. It shows sharp perplexity spikes on hard tasks, indicating instability of learning likely due to task difficulty gaps and ORM degradation. When $\alpha = 0.7$, the agent maintains moderate and slightly decreasing perplexity, suggesting a balanced trade-off and better alignment with the test distribution, leading to more stable training.

D.3 Complexity

Table 8 compares average token usage per tool call and tool-call latency across models. Reasoning-oriented models (e.g., GPT-4o, Qwen3-235B-A22B) consume substantially more tokens and incur higher latency, reflecting their stronger planning and multi-step reasoning behaviors. In contrast, TOOLCAD(Qwen2.5-7B-Instruct) achieves the lowest latency (648 ms) with moderate token usage, showing better efficiency but limited reasoning depth. The increasing token-latency trend suggests a trade-off between reasoning capacity and interaction efficiency in tool-using CAD workflows, where larger reasoning models excel at complex modeling but introduce non-trivial computational overhead.

D.4 Visual Qualitative Results

This section provides complete tool-using agent modeling trajectories for text-to-CAD generation across tasks with varying part counts. The cases include the full tool-calling sequence with model-generated part naming, Boolean Operations (Union/Cut), and detailed CAD construction logs, illustrating intermediate geometry and key modeling details. TOOLCAD’s agent modeling examples are presented in Figure.10.

D.5 Other Analysis

To investigate the performance boundaries of CAD modeling agents, we evaluate scaling behaviors across the Qwen2.5 model family, ranging from 0.5B to 14B parameters. Our analysis focuses

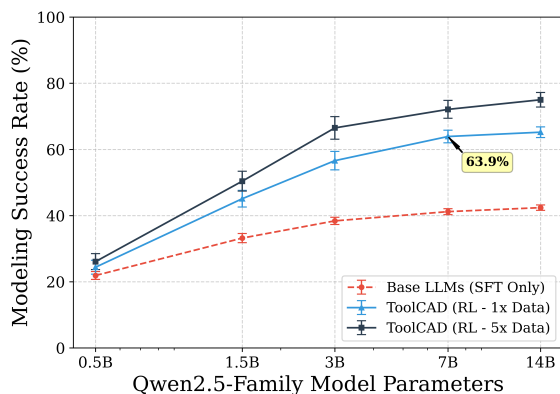


Figure 9: Scaling Laws of CAD Tool-using Agents.

on the trade-off between model parameter scale and online reinforcement learning (RL) exploration scale.

The SFT Performance Plateau. As illustrated in the red dashed curve (see Figure.9), base models via SFT exhibit a rapid saturation effect. While increasing parameters from 0.5B to 3B yields noticeable gains, the performance enters a plateau beyond the 7B scale, with the success rate stagnating. This suggests that traditional supervised scaling is insufficient for mastering the long-horizon tool-call chains required for professional-grade CAD modeling.

RL-Scale as the Primary Performance Driver. Across all model sizes, applying RL consistently outperforms the SFT-only baseline, confirming the importance of RL for structured reasoning and tool-augmented CAD modeling. More critically, increasing online RL training data from 1x to 5x leads to substantially larger gains than simply increasing model parameters. The 5x RL curve shows a clear upward shift across scales, demonstrating that data scaling, rather than model scaling, is the primary bottleneck under the current RL regime. For CAD tool-using agents, online RL data scaling substantially improves modeling success, but the gains are not unbounded—RL training instability throttles the effective scaling range and ultimately bounds the performance ceiling.

Algorithm 1: Online Curriculum RL for CAD Tool-Using Agents

Input: $\mathcal{T}_{\text{pool}}$: Held-out CAD task pool;

π_θ : Policy initialized via \mathcal{M}_{SFT} ;

\mathcal{E} : CAD environment;

\mathcal{V} : Held-in test set;

α : curriculum decay factor ($0 < \alpha < 1$);

N_{update} : policy update interval;

Output: Optimized policy π_θ

Initialize replay buffer $\mathcal{B} \leftarrow \emptyset$;

Group tasks by part-count levels:

$\{\mathcal{T}_1, \dots, \mathcal{T}_L\}$;

Group held-in validation set similarly:

$\{\mathcal{V}_1, \dots, \mathcal{V}_L\}$;

Set curriculum level $\ell \leftarrow 1$;

Initialize running window of recent

perplexities $\mathcal{P}_{\text{recent}} \leftarrow \square$;

while $\ell \leq L$ **do**

 Compute initial perplexity threshold for
 current level;

$$\delta'_\ell = \frac{1}{|\mathcal{V}_\ell|} \sum_{\tau \in \mathcal{V}_\ell} \exp \left(-\frac{1}{|\tau|} \sum_{k=1}^{|\tau|} \log \pi_\theta(a_k | s_k) \right)$$

 Set $\delta \leftarrow \alpha \cdot \delta'_\ell$;

while $\text{mean}(\mathcal{P}_{\text{recent}}) \geq \delta$ **do**

 Sample task $t \sim \mathcal{T}_\ell$;

 Rollout trajectory

$\tau = \{(s_k, a_k, r_k)\} \leftarrow \mathcal{E}(\pi_\theta, t)$;

 Store trajectory in buffer:

$\mathcal{B} \leftarrow \mathcal{B} \cup \{\tau\}$;

 Compute perplexity for τ :

$P(\tau) =$
 $\exp \left(-\frac{1}{|\tau|} \sum_{k=1}^{|\tau|} \log \pi_\theta(a_k | s_k) \right)$;

 Update recent perplexities:

$\mathcal{P}_{\text{recent}} \leftarrow \mathcal{P}_{\text{recent}} \cup \{P(\tau)\}$;

if $|\mathcal{B}| \geq N_{\text{update}}$ **then**

 Update policy π_θ using GRPO
 on \mathcal{B} ;

 Advance to next curriculum level:

$\ell \leftarrow \ell + 1$;

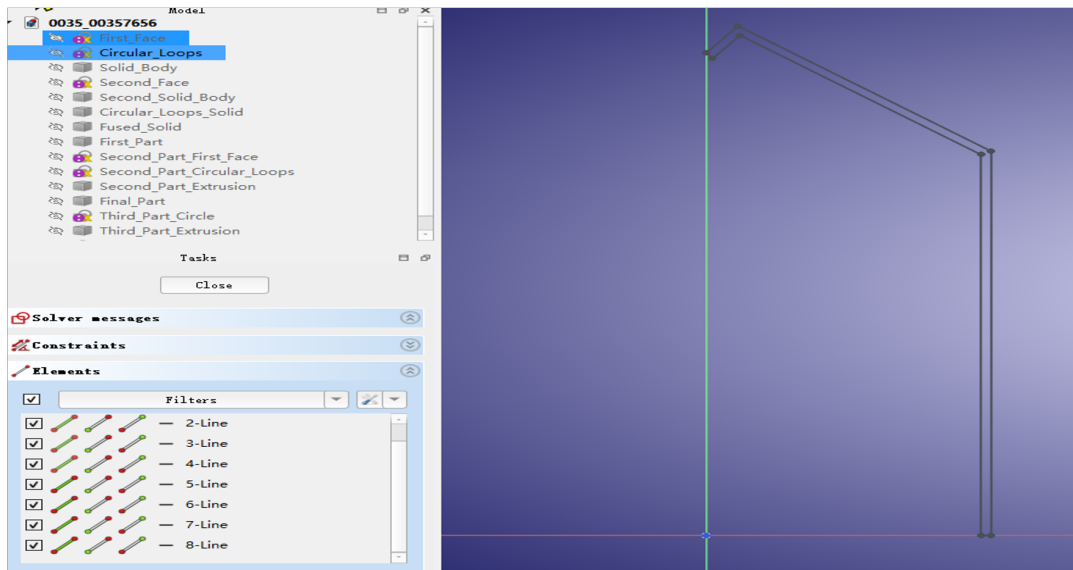
 Reset $\mathcal{P}_{\text{recent}} \leftarrow \square$;

return π_θ

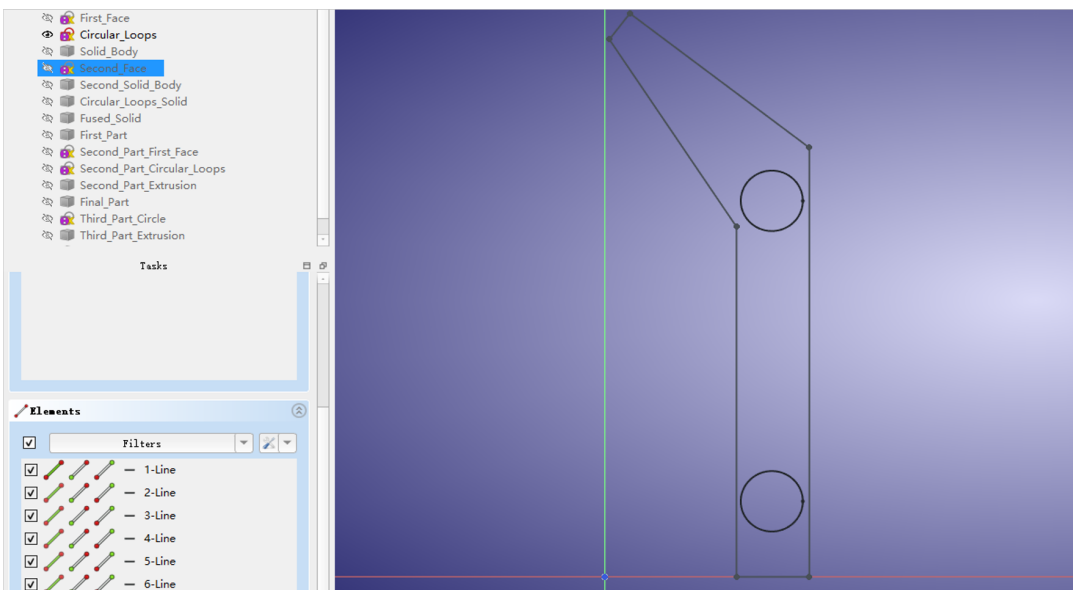
Figure 10: ToolCAD's Agent Modeling Examples

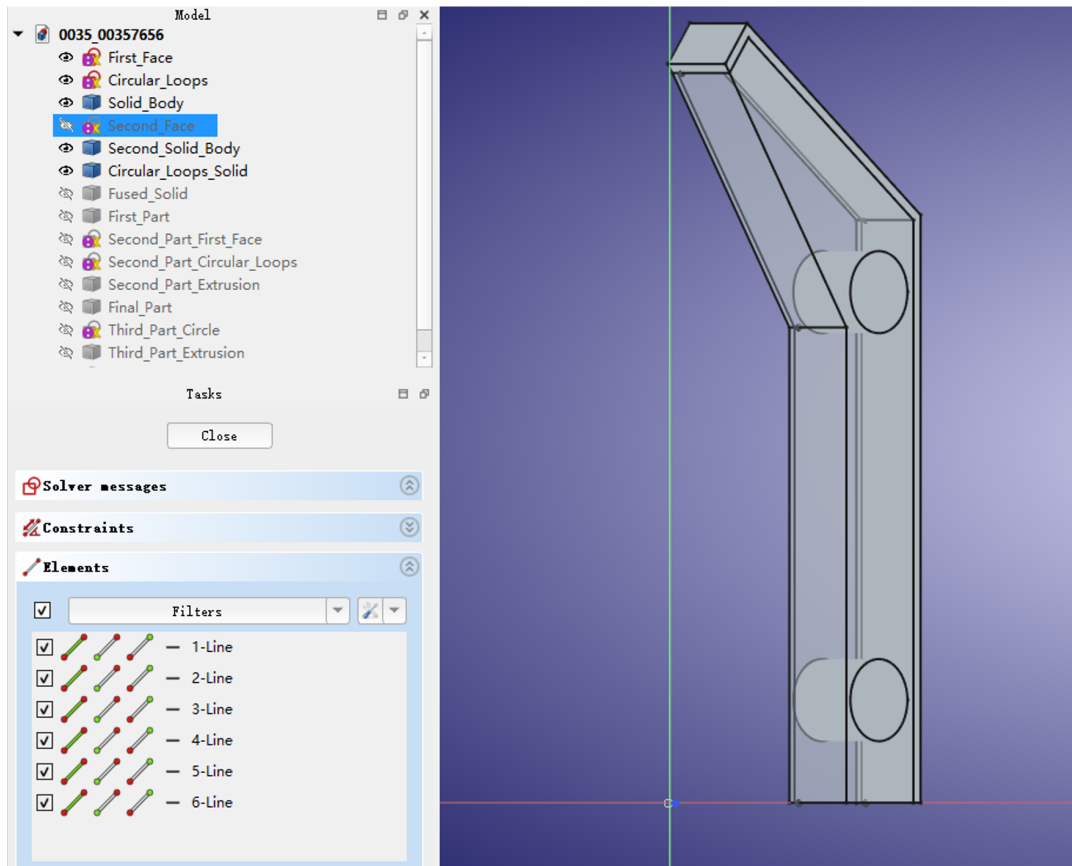


Start by creating a new coordinate system for the first part, setting the Euler angles to 0, 0, and -90 degrees, and the translation vector to 0, 0.0789, and 0. Next, draw the first face using a loop that consists of eight lines. Begin the first line at coordinates 0, 0.7109, and end it at 0.0058, 0.7022. Continue with the second line from 0.0058, 0.7022, to 0.0319, 0.7362. The third line runs from 0.0319, 0.7362, to 0.2603, 0.5607. The fourth line goes from 0.2603, 0.5607, to 0.2603, 0. The fifth line extends from 0.2603, 0, to 0.2702, 0. The sixth line moves from 0.2702, 0, to 0.2702, 0.5656. The seventh line stretches from 0.2702, 0.5656, to 0.03, 0.75. Finally, the eighth line completes the loop by going from 0.03, 0.75, back to the starting point at 0, 0.7109.

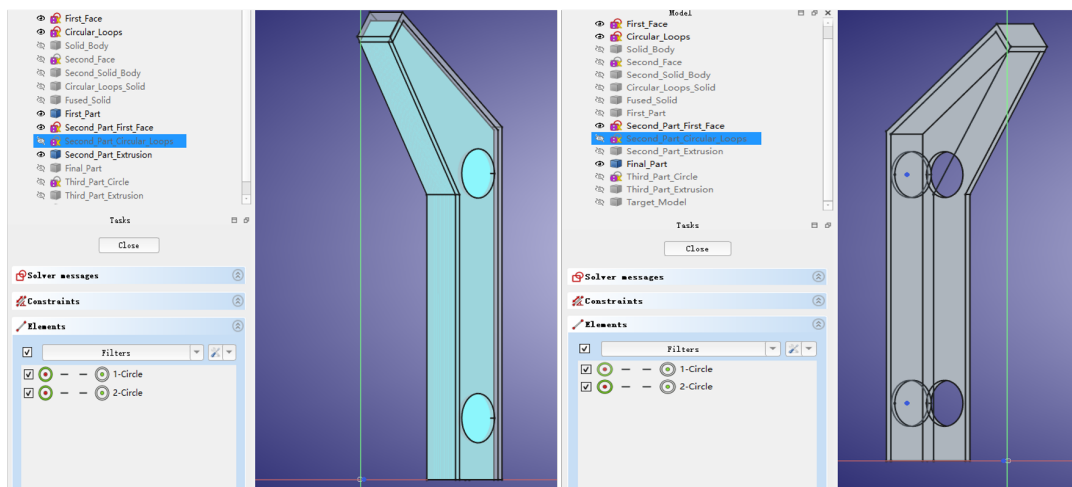


For the second face, create another loop with six lines: The first line starts at 0.0058, 0.7022, and ends at 0.0319, 0.7362. The second line runs from 0.0319, 0.7362, to 0.2603, 0.5607. The third line continues from 0.2603, 0.5607, to 0.2603, 0. The fourth line extends from 0.2603, 0, to 0.1676, 0. The fifth line moves from 0.1676, 0, to 0.1676, 0.4573. The sixth line completes the loop by returning from 0.1676, 0.4573, to 0.0058, 0.7022. Add two circular loops: First circle center: 0.2125, 0.0987, radius 0.0395. Second circle center: 0.2125, 0.4913, radius 0.0395. Extrude the sketch 0.0789 units in the direction of the normal and 0 units in the opposite direction to create a new solid body.



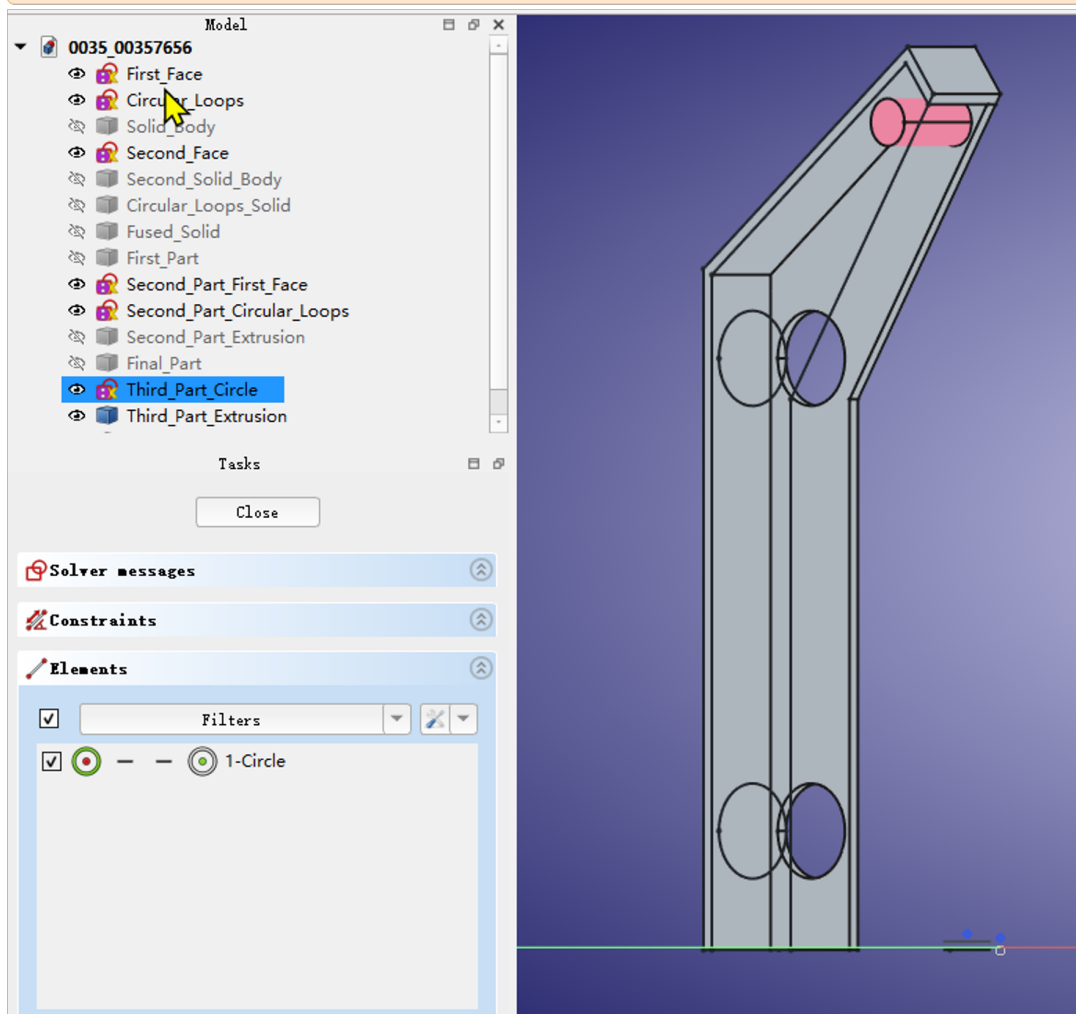


For the second part, create a new coordinate system with Euler angles set to 0, 0, and -90 degrees, and the translation vector set to 0.0058, 0.0789, and 0. Draw the first face using a loop of six lines: First line: 0, 0.7022 to 0.0261, 0.7362 Second: 0.0261, 0.7362 to 0.2546, 0.5607, Third: 0.2546, 0.5607 to 0.2546, 0, Fourth: 0.2546, 0 to 0.1618, 0, Fifth: 0.1618, 0 to 0.1618, 0.4573, Sixth: 0.1618, 0.4573 to 0, 0.7022. Add two circular loops: First circle center: 0.2068, 0.0987, radius 0.0395. Second circle center: 0.2068, 0.4913, radius 0.0395. Extrude the sketch in the normal direction and 0 units in the opposite direction to remove material from the existing body.



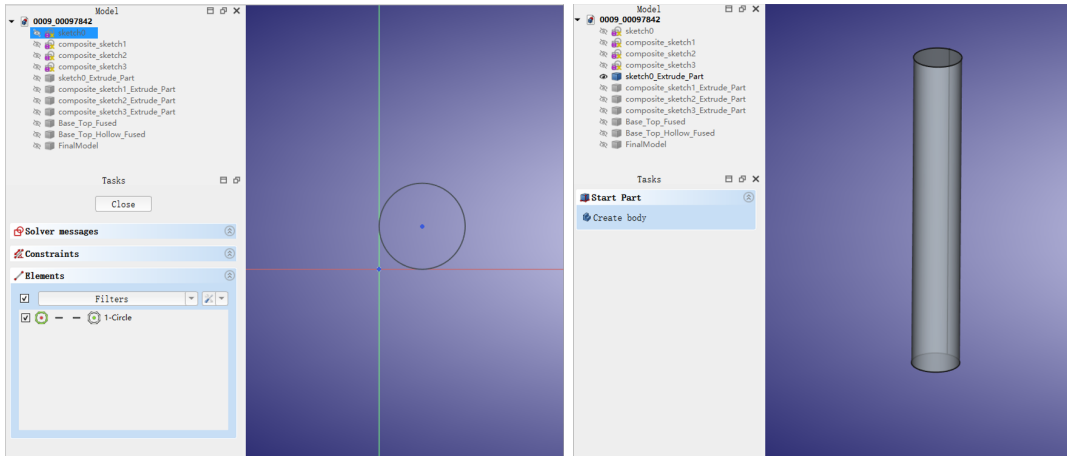


For the third part, create a new coordinate system with Euler angles set to 180 degrees, 0, and -90 degrees, and the translation vector set to 0.0635, 0.0099, and 0.6677. Draw the first face using a single circular loop: Circle center: 0.0197, 0.0197, radius 0.0197. Scale the sketch by 0.0395. Extrude the sketch 0.0789 units in the normal direction and 0 in the opposite direction to add material to the existing body.

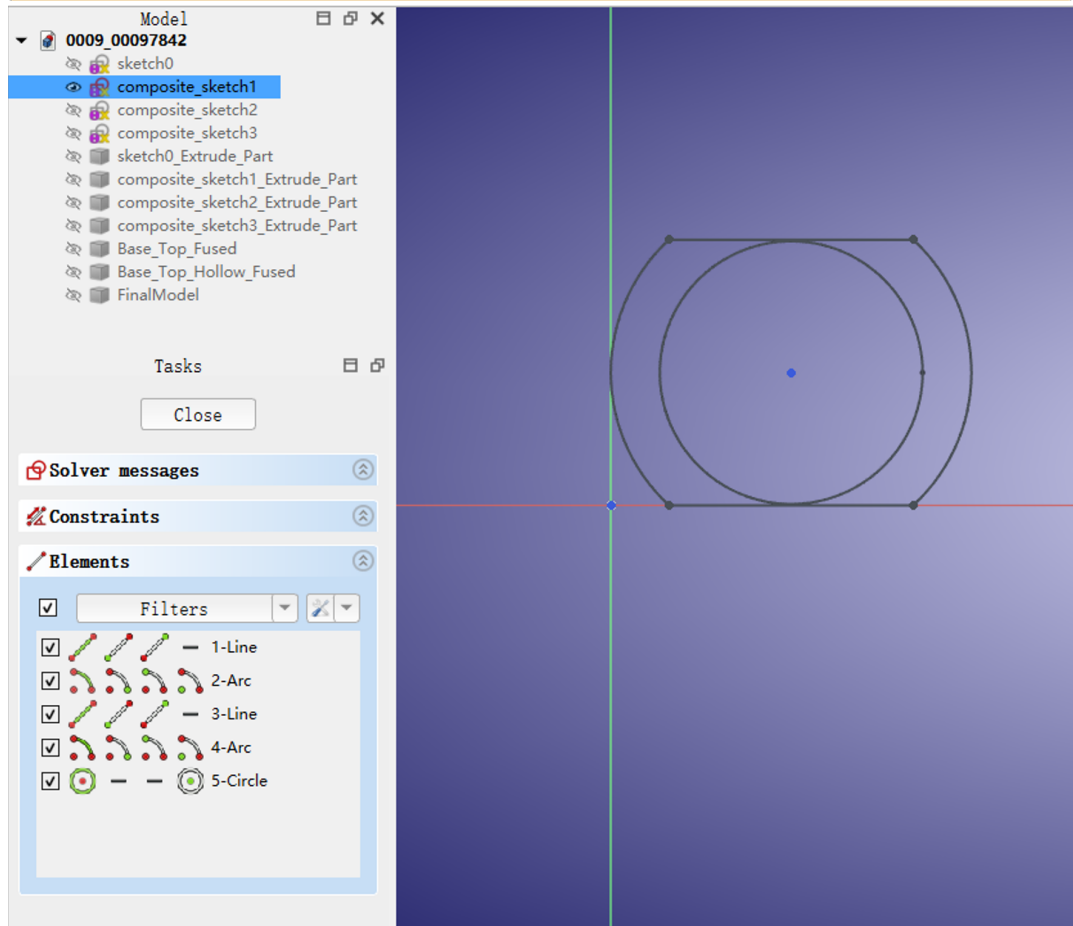


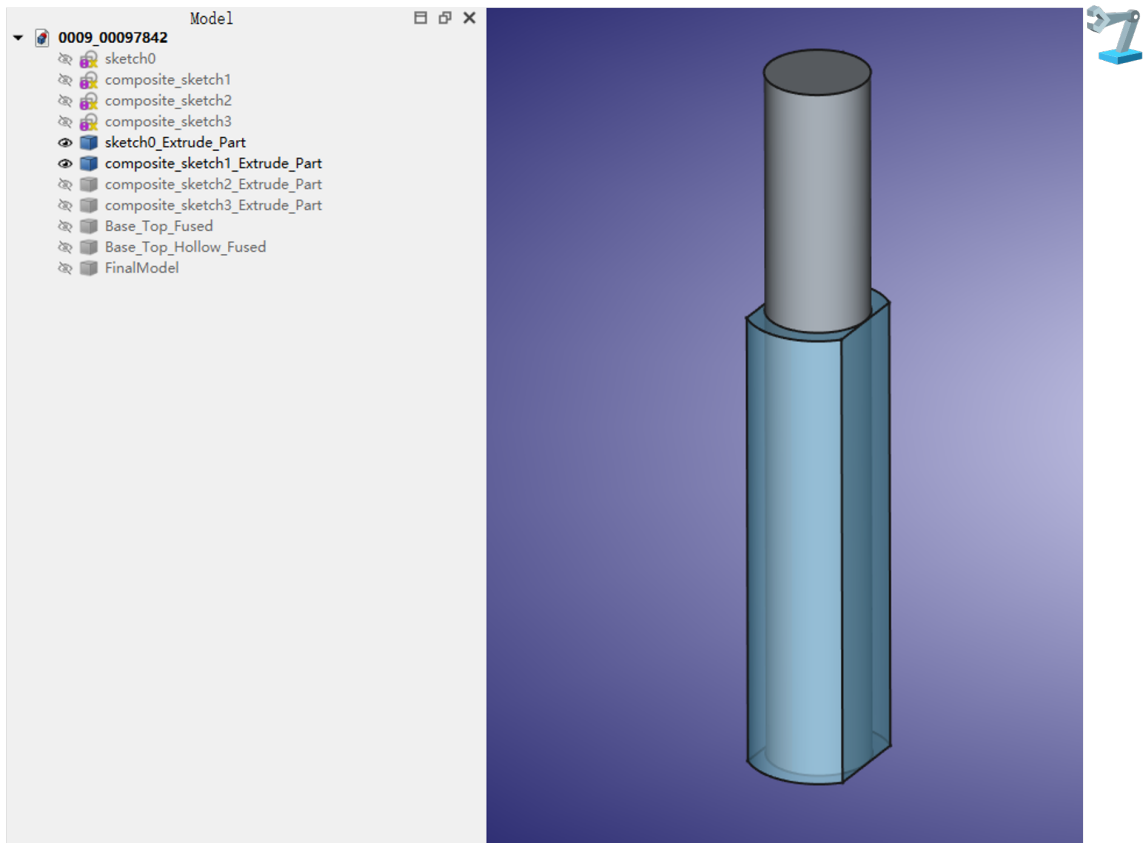


Create a cylindrical base by starting with a new coordinate system set at Euler angles of $[0.0, 0.0, 0.0]$ and a translation vector of $[0.1372, 0.1372, 0.0]$. Draw a 2D sketch on a new face, creating a single loop with a circle centered at $[0.0533, 0.0533]$ and a radius of 0.0533 . Extrude the sketch to a depth of 0.75 units towards the normal, creating a new solid body. The final dimensions of the cylindrical base are length 0.10655134339344864 , width 0.10655134339344864 , and height 0.75 .

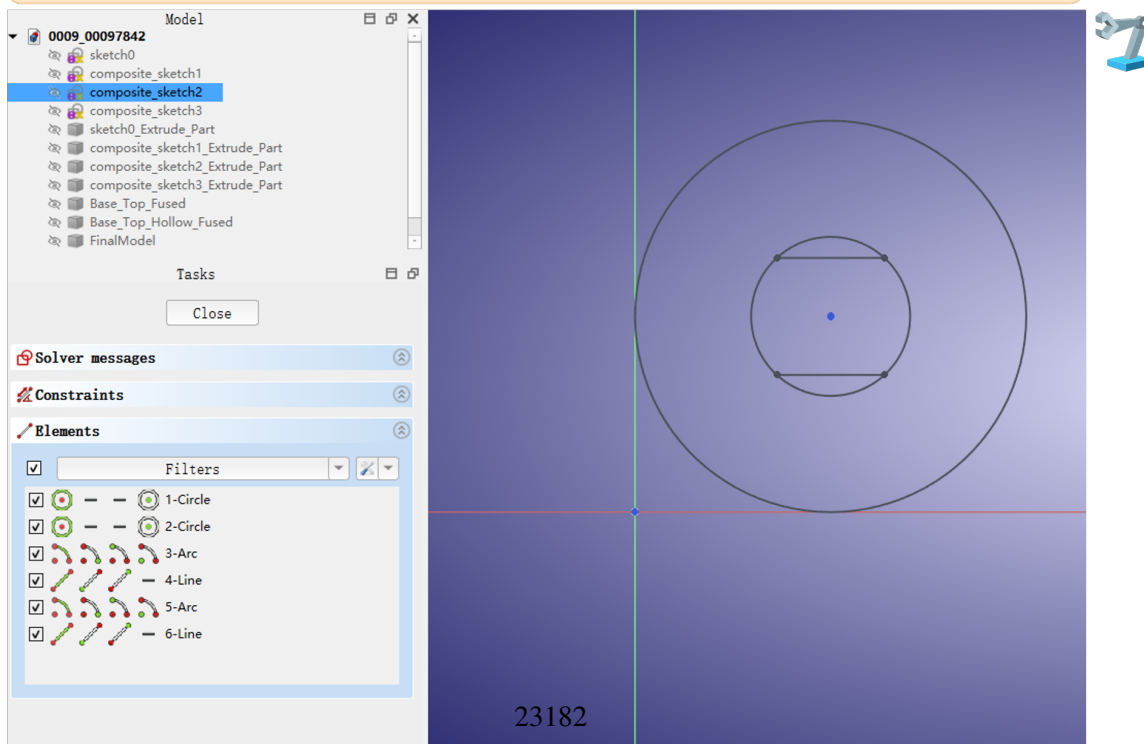


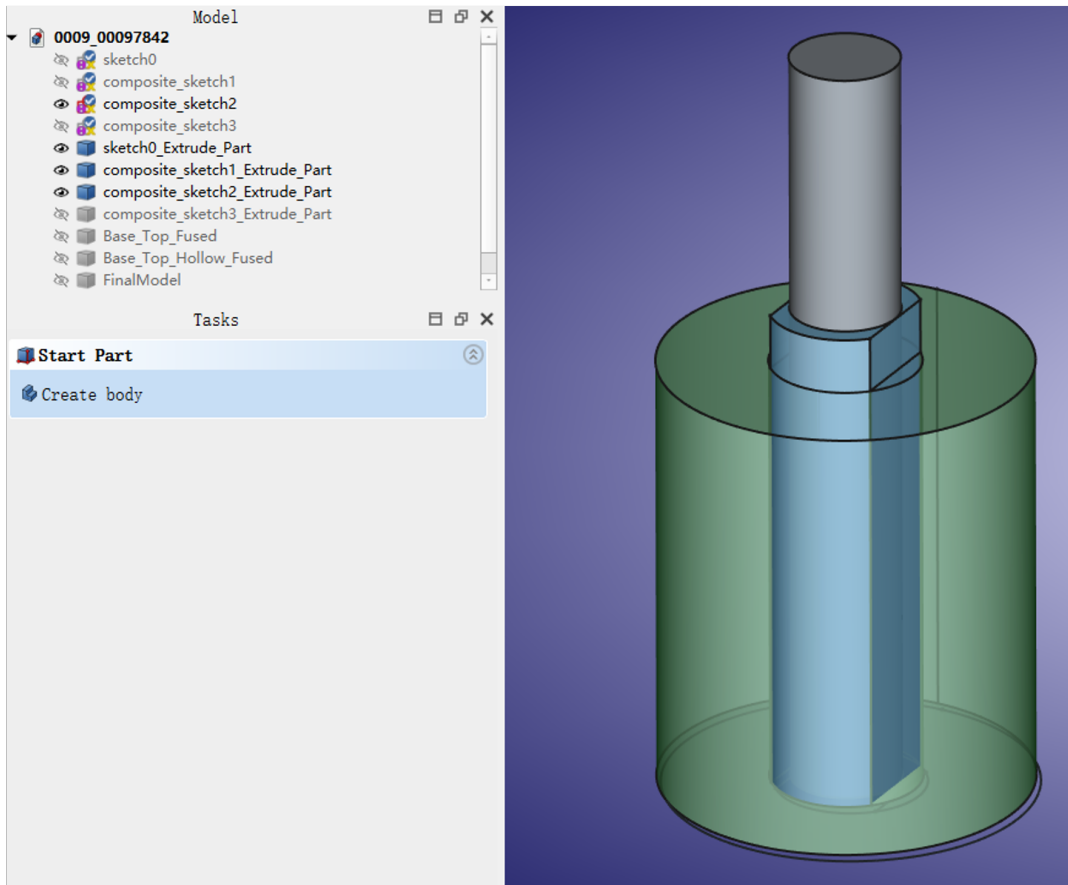
Next, construct a smaller cylindrical top. Begin by setting up a new coordinate system with Euler angles of $[0.0, 0.0, 0.0]$ and a translation vector of $[0.1173, 0.1366, 0.0]$. On a new face, draw a 2D sketch with two loops. The first loop consists of a line from $[0.0236, 0.0]$ to $[0.1227, 0.0]$, an arc from $[0.1227, 0.0]$ to $[0.1227, 0.1077]$ with a midpoint at $[0.1463, 0.0538]$, a line from $[0.1227, 0.1077]$ to $[0.0236, 0.1077]$, and an arc from $[0.0236, 0.1077]$ to $[0.0236, 0.0]$ with a midpoint at $[0.0, 0.0538]$. The second loop is a circle centered at $[0.0732, 0.0538]$ with a radius of 0.0533 . Extrude the sketch to a depth of 0.488 units towards the normal, joining it to the existing body. The final dimensions of the smaller cylindrical top are length 0.1463010673536989 , width 0.10765550239234448 , and height 0.48803827751196166 .



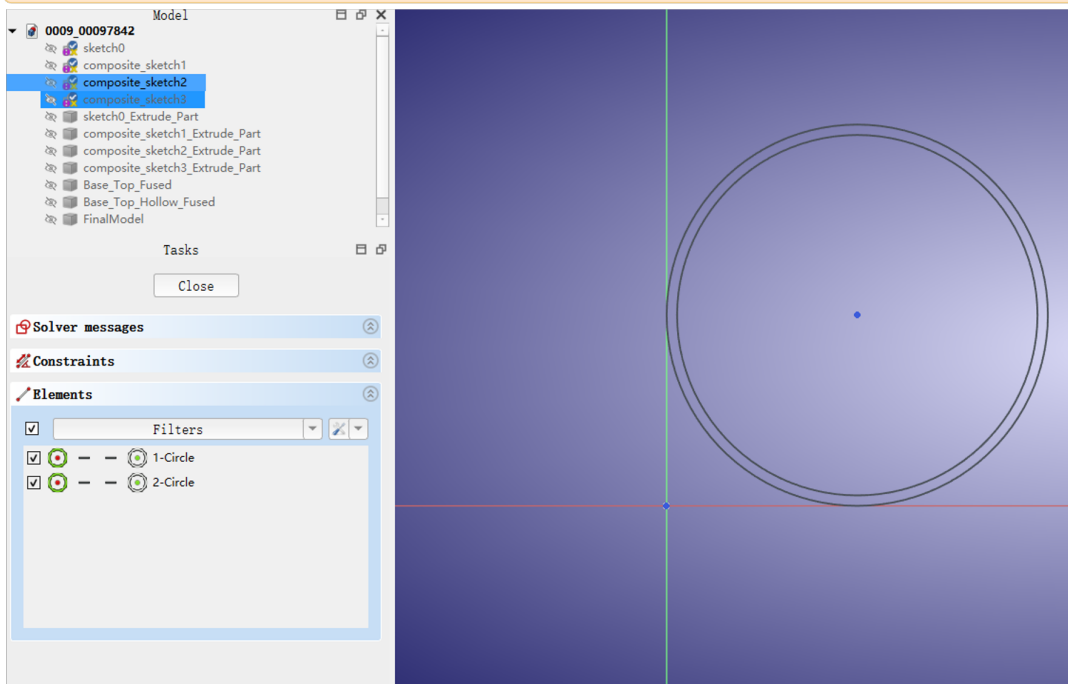


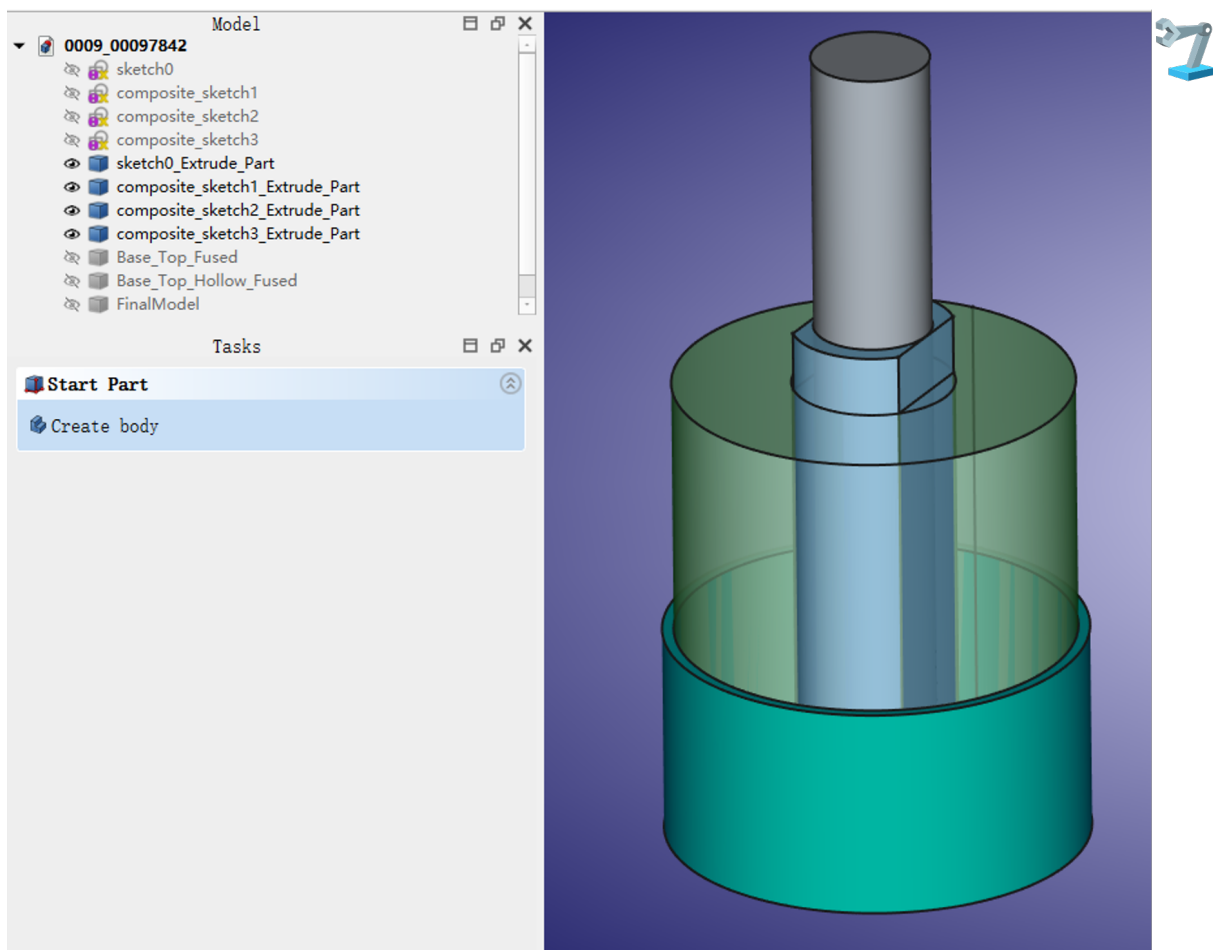
Construct a hollow cylindrical section by setting up a new coordinate system with Euler angles of $[0.0, 0.0, 0.0]$ and a translation vector of $[0.0104, 0.0104, 0.0]$. On a new face, draw a 2D sketch with three faces. The first face has two loops: the first loop is a circle centered at $[0.1801, 0.1801]$ with a radius of 0.1801 , and the second loop is a circle centered at $[0.1801, 0.1801]$ with a radius of 0.0732 . The second face consists of a single loop with an arc from $[0.1306, 0.1263]$ to $[0.2296, 0.1263]$ with a midpoint at $[0.1801, 0.107]$ and a line from $[0.2296, 0.1263]$ to $[0.1306, 0.1263]$. The third face consists of a single loop with an arc from $[0.1306, 0.2339]$ to $[0.2296, 0.2339]$ with a midpoint at $[0.1801, 0.2533]$ and a line from $[0.2296, 0.2339]$ to $[0.1306, 0.2339]$. Extrude the sketch to a depth of 0.4331 units towards the normal, joining it to the existing body. The final dimensions of the hollow cylindrical section are length 0.3602318733897681 , width 0.3602318733897681 , and height 0.43310636731689356 .





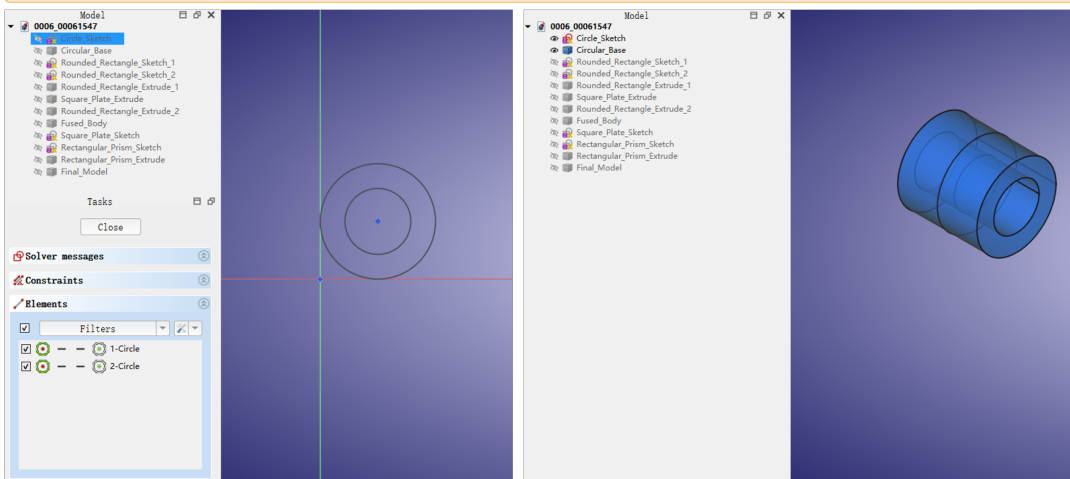
Finally, construct a circular cap by setting up a new coordinate system with Euler angles of $[0.0, 0.0, 0.0]$ and a translation vector of $[0.0, 0.0, 0.0]$. On a new face, draw a 2D sketch with two loops: the first loop is a circle centered at $[0.1905, 0.1905]$ with a radius of 0.1905 , and the second loop is a circle centered at $[0.1905, 0.1905]$ with a radius of 0.1801 . Extrude the sketch to a depth of 0.1932 units towards the normal, joining it to the existing body. The final dimensions of the circular cap are length 0.380934854619065 , width 0.380934854619065 , and height 0.19322782480677214 .



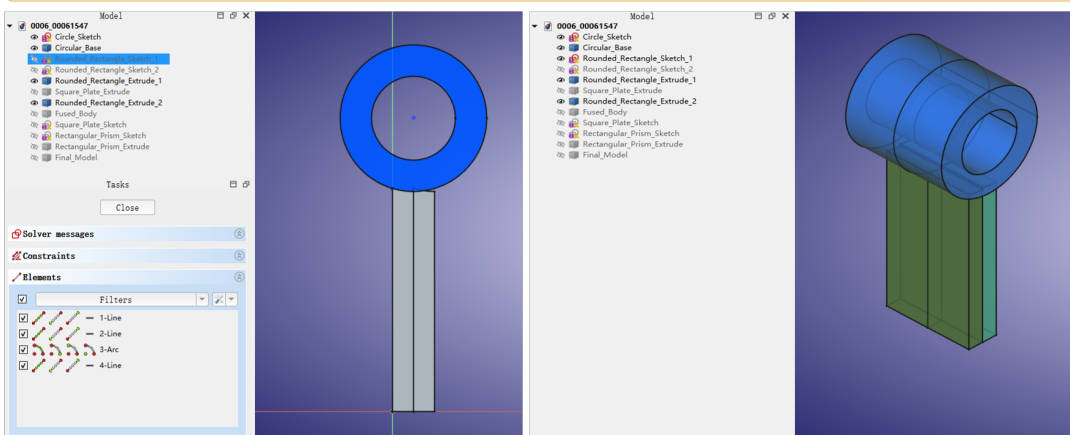




Create a new coordinate system for the first part by setting the Euler angles to $[0.0, 0.0, -90.0]$ and the translation vector to $[0.3214, 0.3, 0.45]$. On the first face, draw a circle with a center at $[0.15, 0.15]$ and a radius of 0.15 . Inside this circle, draw a smaller circle with the same center and a radius of 0.0857 . Extrude the sketch 0.15 units in both directions along the normal to create a solid body.

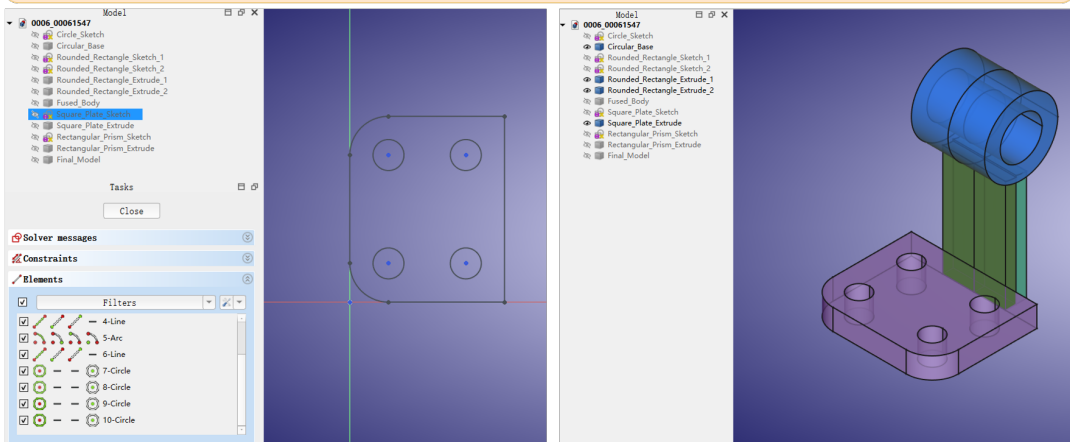


For the second part, create another coordinate system with Euler angles set to $[0.0, 0.0, -90.0]$ and a translation vector of $[0.4286, 0.3, 0.0]$. On the first face, draw a rectangular shape with rounded edges. Start by drawing a line from $[0.0, 0.0]$ to $[0.0429, 0.0]$, then from $[0.0429, 0.0]$ to $[0.0429, 0.45]$, followed by an arc from $[0.0429, 0.45]$ to $[0.0, 0.4563]$ with a midpoint at $[0.0212, 0.4516]$, and finally a line from $[0.0, 0.4563]$ back to $[0.0, 0.0]$. On the second face, draw a similar rectangular shape but with the starting point at $[0.0429, 0.0]$. Extrude the sketch 0.1286 units in both directions along the normal to add it to the existing body.

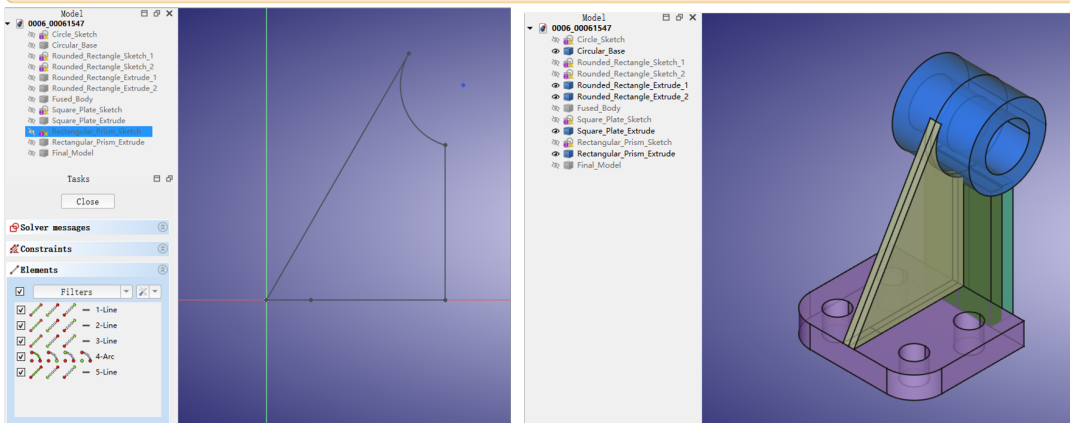




For the third part, create a new coordinate system with Euler angles set to $[0.0, 0.0, 180.0]$ and a translation vector of $[0.0, 0.5571, 0.0]$. On the first face, draw a square plate with rounded corners and four circular holes. Start by drawing an arc from $[0.0, 0.1071]$ to $[0.1071, 0.0]$ with a midpoint at $[0.0314, 0.0314]$, then draw a line from $[0.1071, 0.0]$ to $[0.4286, 0.0]$, followed by a line from $[0.4286, 0.0]$ to $[0.4286, 0.5143]$, another line from $[0.4286, 0.5143]$ to $[0.1071, 0.5143]$, an arc from $[0.1071, 0.5143]$ to $[0.0, 0.4071]$ with a midpoint at $[0.0314, 0.4829]$, and finally a line from $[0.0, 0.4071]$ back to $[0.0, 0.1071]$. Inside the square, draw four circles with centers at $[0.1071, 0.1071]$, $[0.1071, 0.4071]$, $[0.3214, 0.1071]$, and $[0.3214, 0.4071]$, each with a radius of 0.0429 . On the second face, draw a rectangular shape. Extrude the sketch 0.0857 units in the opposite direction of the normal to add it to the existing body.



For the fourth part, create a new coordinate system with Euler angles set to $[0.0, 0.0, -90.0]$ and a translation vector of $[0.0, 0.3, 0.0857]$. On the first face, draw a rectangular prism with a cylindrical section. Start by drawing a line from $[0.0, 0.0]$ to $[0.1071, 0.0]$, then from $[0.1071, 0.0]$ to $[0.4286, 0.0]$, followed by a line from $[0.4286, 0.0]$ to $[0.4286, 0.3705]$, an arc from $[0.4286, 0.3705]$ to $[0.3416, 0.5895]$ with a midpoint at $[0.332, 0.4589]$, and finally a line from $[0.3416, 0.5895]$ back to $[0.0, 0.0]$. Extrude the sketch 0.0214 units in both directions along the normal to add it to the existing body.



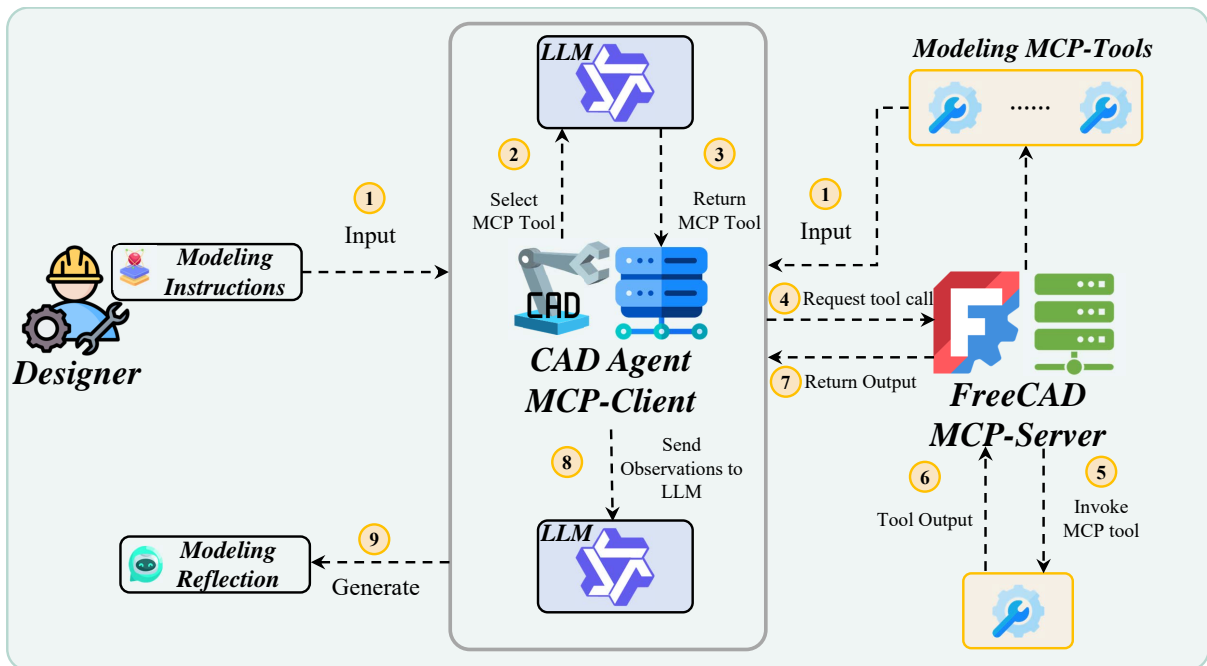


Figure 11: Integration of MCP provides structured environment and tool management, which enhances the extensibility and flexibility of CAD modeling tools.

Figure 12: feedback of create_complex_sketch

```

1 def create_complex_sketch(elements: List[SketchElement] = None, sketch_name: str =
2   None) -> InterfaceResult:
3
4   """
5   Creates a composite sketch consisting of multiple geometric elements.
6   This function is used to batch-draw sketch profiles in FreeCAD composed of lines
7   , circles, arcs, or splines.
8   Args:
9     elements (List[SketchElement]): A list of geometric elements
10    sketch_name (str): Name of the sketch.
11   """
12   # Call FreeCAD API to draw elements
13   ...
14   # Return interface result
15   # 1. Successfully created sketch and derived face.
16   return InterfaceResult(True, [Action(description='Successfully created sketch'+
17     sketch_name+'and its sketch-derived face'+face_name)])
18   # 2. Sketch creation failed due to geometric issues, suggest step-by-step
19   creation
20   return InterfaceResult(False, [Action(description='Sketch creation failed: '+
21     feedback+'. Please try creating each profile loop one by one.')])
22   # 3. Execution error occurred during sketch generation (include exception
23   message)
24   return InterfaceResult(False, [Action(description='Sketch creation failed due to
25     an internal error:'+str(error_message))])

```

Figure 13: feedback of bool_operation

```
1 def boolean_operation(  
2     base_object_name: str,  
3     tool_object_name: str,  
4     operation: Literal["cut", "fuse", "common"],  
5     name: str = None,  
6 ) -> InterfaceResult:  
7     """  
8     Performs a boolean operation (cut, fuse, or common) between two solid objects to  
9         create a new 3D model entity.  
10  
11     This function executes the specified boolean operation between the base object  
12     and the tool object based on the given operation type:  
13     - "cut": subtracts the tool object from the base object;  
14     - "fuse": merges the base and tool objects;  
15     - "common": computes the intersection of the base and tool objects.  
16  
17     Args:  
18         base_object_name (str): The name of the base object involved in the boolean  
19         operation.  
20         tool_object_name (str): The name of the tool object used in the boolean  
21         operation.  
22         operation (Literal["cut", "fuse", "common"]): The type of boolean operation  
23         to perform.  
24         name (str): The name of the resulting object.  
25     """  
26     # Call FreeCAD API to perform Boolean operations.  
27     ...  
28     # Return interface result  
29     # 1. Successfully created a new solid as a result of the Boolean operation.  
30     return InterfaceResult(True, [Action(description='A new solid'+name+' was  
31         created by performing the Boolean operation'+operation+'.')])  
32     # 2. Boolean operation failed; include operation type, object names, and error  
33     message.  
34     return InterfaceResult(False, [Action(description='The Boolean operation '+  
35         operation+' between base object '+base_object_name+' and tool object '+  
36         tool_object_name+' failed. Error: {str(e)}")])  
37 ])
```