COMACT: COM AS A UNIFIED ACTION SPACE FOR PROFESSIONAL SOFTWARE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Current computer-use agents remain fundamentally limited on professional software. GUI-based agents struggle with fragile visual grounding and error accumulation over long action sequences, while MCP- and API-based agents require programmable interfaces that closed-source commercial software rarely provides. We observe that the **Component Object Model (COM)**, a Windows-native interface widely supported by commercial professional applications, provides programmatic access to their full functionality through a uniform invocation mechanism. We therefore present **ComAct**, which uses COM as a unified action space across professional applications, replacing long, fragile GUI actions with compact executable COM programs. We stress-test ComAct on industrial CAD and build **ComCADBench**, comprising 1,000 tasks across three mainstream CAD applications, spanning standalone design tasks and multi-stage engineering workflows. Frontier models fail almost completely under GUI interaction but achieve non-trivial success when switched to ComAct, and the same switch also yields consistent zero-shot improvements on Office, Adobe, and OS-shell tasks. Yet off-the-shelf models still exhibit a large gap between executable code and task-correct outputs. To close this gap, we build **ComForge**, a scalable platform of parallel real computer environments for large-scale training, and train a self-correcting agent through a three-stage progressive framework with continuous geometric rewards. Experiments show our agent achieves state-of-the-art performance on ComCADBench, particularly on long-horizon workflows, and generalizes well to external CAD benchmarks. The environment, code, data, and models will be publicly released.

1 INTRODUCTION

Computer-use agents have made remarkable progress on general-purpose tasks such as web browsing and document editing (Anthropic, 2024; OpenAI, 2025). However, on the professional software that powers engineering and design, they remain far from reliable. Current agents mainly follow two paradigms: GUI-based interaction and MCP-/API-based invocation. GUI-based agents act through clicks and keystrokes grounded in screenshots (OpenAI, 2025). They offer broad accessibility but rely on fine-grained visual grounding, which is fragile in visually dense professional graphic interfaces (Li et al., 2025b). Besides, professional workflows can require hundreds of sequential GUI actions where small per-step errors can easily compound into failure. MCP-/API-based agents act through programmatic calls (Qin et al., 2024), which provide structured control but are constrained by fragmented, application-specific interfaces and limited public API support in closed-source commercial software. As a result, neither paradigm delivers both the reliability and universality required for real-world professional workflows.

This trade-off calls for a unified interface that combines the universality of GUI interaction with the reliability of programmatic control. We identify the Component Object Model (COM) (Microsoft, 2024), a Windows-native interoperability framework, as a promising solution. As shown in Figure 1, COM exposes rich software functionality through structured object interfaces with callable methods and properties. It is widely supported by commercial professional software, including SolidWorks, Microsoft Office, Autodesk, and Adobe applications. Operations accessible through GUI interaction can be executed directly through COM calls, while the same object-method-property invocation pattern provides a common mechanism for interacting with diverse applications. Thus, a compact

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.

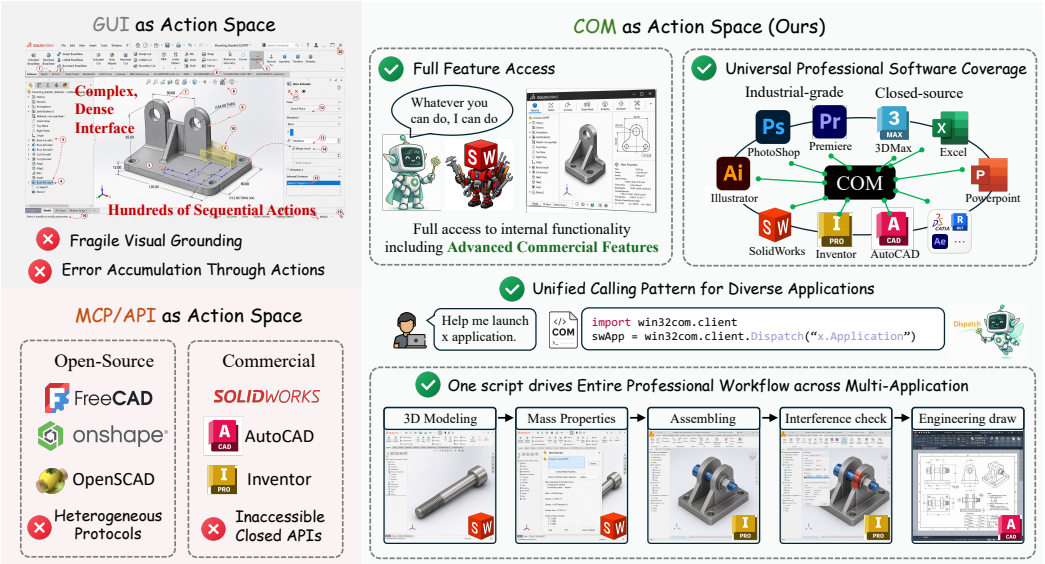

Figure 1: Comparison of existing computer-use paradigms and ComAct. GUI-based agents rely on fragile visual grounding and suffer from long-horizon error accumulation, while API-based agents are constrained by fragmented and limited interfaces. In contrast, we leverage COM as a unified semantic programmatic interface, enabling executable program synthesis and cross-application workflows.

COM program can replace dozens of sequential GUI actions and complete an entire workflow. We therefore propose ComAct, which uses COM as a unified action space across professional applications. Agents directly manipulate semantic software objects through generated COM programs, while reserving visual perception for coarse-grained observation and verification.

To validate ComAct in a demanding professional setting, we choose industrial Computer-Aided Design (CAD) as our primary domain. CAD presents particularly challenging interaction requirements, including dense graphical interfaces, continuous 3D spatial reasoning, strict numerical precision, and long sequences of causally dependent engineering operations. To systematically evaluate agents in this setting, we construct ComCADBench, a benchmark of 1,000 tasks across SolidWorks, Autodesk Inventor, and AutoCAD, covering seven engineering activities and both standalone tasks and multi-stage workflows. Rather than evaluating intermediate actions, ComCADBench directly verifies the resulting software artifacts, enabling different interaction paradigms to be compared under the same task-level criterion.

Experiments reveal a substantial interaction gap: frontier models achieve near-zero success under GUI interaction, even with enhanced visual grounding, but obtain non-trivial performance once switched to ComAct. The same switch also yields consistent zero-shot improvements on Office, Adobe, and OS-shell tasks, demonstrating the generality of the ComAct paradigm beyond CAD.

Despite these improvements, off-the-shelf models still exhibit a large gap between executable code and task-correct outputs. To close this gap, we build ComForge, a scalable platform for large-scale training and evaluation in parallel real professional software environments, and use it to train a self-correcting 9B agent through a progressive three-stage framework. The first stage learns the mapping from instructions to COM programs; the second introduces multi-turn refinement; and the third performs task-level optimization with continuous geometric rewards. The resulting agent substantially improves over off-the-shelf models, particularly on long-horizon workflows, and generalizes to external CAD benchmarks.

Our contributions are summarized as follows:

- We propose **ComAct**, which uses COM as a unified action space for professional software agents.
- We construct **ComCADBench**, a benchmark of 1,000 CAD tasks across three mainstream CAD applications, validating the effectiveness of the ComAct paradigm.

- We build **ComForge**, a scalable platform of parallel real Windows environments for large-scale training and evaluation.
- We train a **self-correcting ComAct agent** through progressive three-stage training, achieving strong performance on ComCADBench and generalization to external CAD benchmarks.

2 Related Work

2.1 Computer-use Agents

Existing computer-use agents mainly follow two paradigms: GUI-based and API-based interaction. GUI-based agents (OpenAI, 2025; Cheng et al., 2024; Agashe et al., 2024; Qin et al., 2025; Zhang et al., 2025a; Hong et al., 2024; Zhang et al., 2025b; Yang et al., 2025; Zheng et al., 2024), powered by MLLMs, rely on screenshot-based visual grounding and low-level mouse-and-keyboard control, achieving strong performance on general desktop tasks but remaining fragile in visually dense professional software environments. Programmatic agents (Qin et al., 2024; Anthropic, 2024; Zhang et al., 2024; Hu et al., 2025; Song et al., 2025b; Lai et al., 2024; Wu et al., 2024; Wang et al., 2024; Song et al., 2025a) instead execute commands or APIs directly, providing more reliable execution but relying on fragmented, application-specific interfaces with limited interoperability across software ecosystems. In contrast, our work introduces COM as a unified, code-centric action space that combines the reliability of programmatic execution with the broad functional coverage of GUI interaction, while reserving visual perception for coarse-grained observation and verification.

2.2 CAD Generation

Existing CAD generation methods mainly fall into two categories: sequence generation and code generation. Sequence-based approaches (Seff et al., 2020; Willis et al., 2021; Dupont et al., 2024; Li et al., 2025a; Zhang et al., 2025c) such as DeepCAD (Wu et al., 2021) and Text2CAD (Khan et al., 2024) model CAD construction as parametric command sequences, while code-based approaches including CAD-Coder (Guan et al., 2026) and CAD-Recode (Rukhovich et al., 2025) generate executable CAD scripts in libraries such as CadQuery and build123d. Although these methods demonstrate promising capabilities for geometric modeling, they remain limited to constrained operation vocabularies or lightweight geometric libraries, lacking support for complex industrial workflows and advanced functionalities in professional CAD software. In contrast, our work directly treats industrial CAD software as the execution environment through COM-based interaction, enabling agents to manipulate commercial CAD applications and operate across diverse engineering workflows.

3 ComAct: COM as a Unified Action Space

Component Object Model. COM is a Windows-native interface standard for programmatic communication with desktop applications. A COM-enabled application exposes its functionality as a hierarchy of objects with callable methods and properties, accessible from languages with COM bindings such as Python via `win32com.client`. COM is widely supported by commercial professional software, including SolidWorks, Office, Autodesk and Adobe applications. Operations accessible through the GUI can be executed directly through COM calls, while the same object-method-property invocation pattern provides a common mechanism for cross-application communication. Thus, a single COM program can replace dozens of sequential GUI actions and complete an entire workflow. This combination of broad coverage, full functionality and uniform invocation allows COM to serve as a unified action space for professional software manipulation.

Formulation of ComAct. We propose ComAct, which uses COM as a unified action space. Specifically, we formalize professional software manipulation as a partially observable Markov decision process $\langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \Omega \rangle$, with transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and observation function $\Omega : \mathcal{S} \to \mathcal{O}$, where $\mathcal{S}, \mathcal{O}, \mathcal{A}$ denote the state, observation, and action spaces, respectively. Each action $a \in \mathcal{A}$ is an executable Python script invoking COM interfaces rather than atomic GUI operations. Given a natural language instruction $\mathcal{I}$ and initial state s_0, the agent iteratively receives an

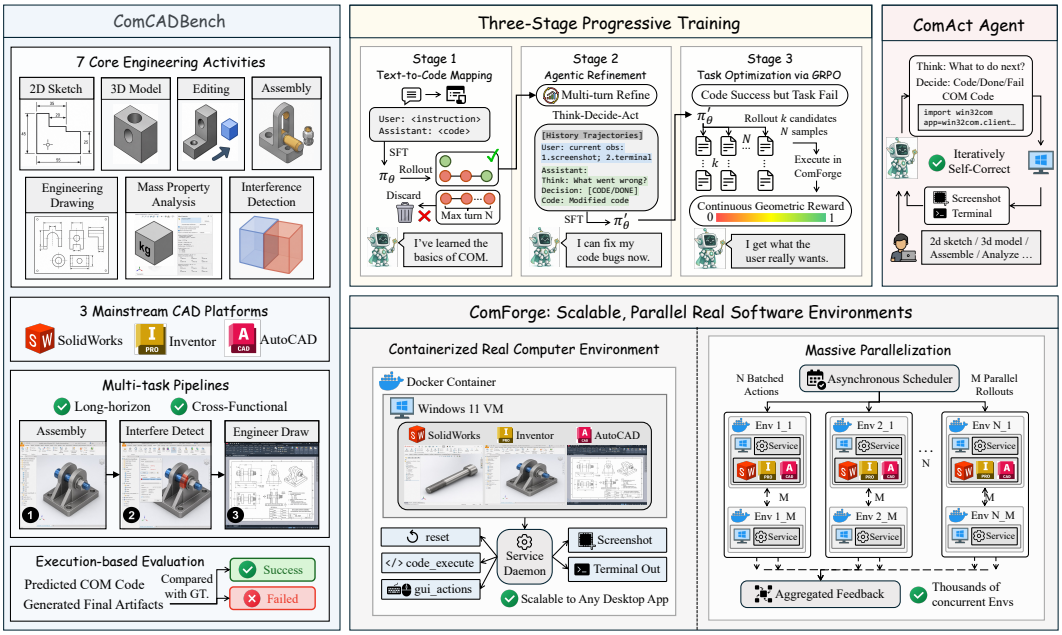

Figure 2: Overview of ComAct: three-stage progressive training (text-to-code SFT, agentic SFT, and GRPO with continuous geometric reward) of the ComAct agent; ComCADBench, which compares generated artifacts against ground truth in real CAD software; and ComForge, supporting 1000+ parallel real Windows environments for training and evaluation.

observation $o_t = \Omega(s_t) \in \mathcal{O}$, comprising a screenshot and execution feedback, and predicts a decision token $\delta_t \in \{\texttt{CODE}, \texttt{DONE}, \texttt{FAIL}\}$. If $\delta_t = \texttt{CODE}$, it generates and executes a script $a_t \in \mathcal{A}$ to transition the state; otherwise, or once a step budget is reached, the episode terminates. The resulting trajectory is evaluated by whether the final software artifact satisfies $\mathcal{I}$.

4 ComCADBench: Evaluating Agent Paradigms in CAD Software

To validate ComAct in a demanding professional setting, we choose industrial CAD as our primary domain, which requires precise geometric reasoning, strict operation dependencies, and long-horizon workflows in professional applications. We therefore construct ComCADBench, a benchmark for evaluating agents operating real CAD software through different interaction paradigms.

Benchmark Construction. Tasks are sampled from three established datasets of real human CAD models: Text2CAD (Khan et al., 2024), SketchGraphs (Seff et al., 2020), and Fusion360 Gallery (Willis et al., 2021). Following their official splits, we construct the benchmark from held-out test samples while using training splits for agent training. Each task is paired with a natural-language instruction, a ground-truth final artifact, and a reference COM script verified in ComForge (Section 5) to confirm feasibility. We further derive downstream tasks such as engineering drawing and interference detection to capture multi-stage engineering workflows.

Table 1: Statistics of ComCADBench.

Statistics	Avg / Max / Min
Instruction (tokens)	
Single-task	578.8 / 1425 / 204
Multi-task	481.5 / 1108 / 274
Overall	523.2 / 1425 / 204
Entities per design	12.7 / 16 / 10
GT script (LoC)	
Single-task	135.4 / 382 / 65
Multi-task	176.3 / 339 / 109
Overall	156.1 / 382 / 65
COM API calls	
Single-task	22.8 / 39 / 7
Multi-task	33.7 / 49 / 12
Overall	29.3 / 49 / 7

Statistics. The final benchmark comprises 1,000 tasks across three mainstream CAD applications, SolidWorks, Autodesk Inventor, and AutoCAD, and spans seven activities: 3D modeling, 2D sketching, editing, assembly, engineering drawing, mass-property analysis, and interference detection.

Tasks fall into two levels: 400 single CAD design tasks and 600 multi-stage engineering workflows. Table 1 highlights their difficulty and diversity: instructions run to hundreds of tokens, and a single solution often coordinates dozens of COM calls over more than a hundred lines of code. Detailed construction, statistics, and samples are provided in Appendix A.

5 COMFORGE: SCALABLE REAL PROFESSIONAL SOFTWARE ENVIRONMENTS

Training agents for professional software requires execution within authentic computer environments at massive scale. However, existing agent environments are rarely built to host heavy commercial Windows applications at training scale. In this section, we build ComForge, a scalable platform of real computer environments that is architecturally general and extensible to other desktop software. It serves as the execution backend for both ComCADBench and our training pipeline.

Containerized Environments and Unified Interface. As shown in Figure 2, each ComForge environment is packaged as a Docker container hosting a full Windows virtual machine with target applications pre-installed. A lightweight service daemon exposes a unified interface of three action types: `code_execute` for executing COM-based Python programs, `gui_action` for mouse and keyboard interaction, and `reset` for restoring the initial state. After each action, the daemon returns the current screenshot with execution outputs as the observation, abstracting heterogeneous software instances into a homogeneous request-response interface.

Massive Parallelization and Training Integration. Professional software operations incur substantial execution latency, making serial trajectory collection a major bottleneck for RL training. ComForge addresses this with an asynchronous scheduler that dynamically dispatches batched requests to idle environments and aggregates observations in parallel, scaling to thousands of concurrent software instances. It also computes artifact-level rewards by exporting the resulting designs and comparing them against the ground truth. We integrate ComForge into the ms-swift (Zhao et al., 2024) GRPO pipeline, where it handles action execution and reward computation throughout training.

6 PROGRESSIVE TRAINING FOR SELF-CORRECTING COMACT AGENTS

Building on ComAct and ComForge, we train a self-correcting agent that translates natural-language instructions into executable COM programs and iteratively refines them through environment interaction until task completion. As illustrated in Figure 2, we adopt a three-stage progressive training framework, where each stage targets the main failure mode exposed by the previous one. All training data are derived exclusively from the official training splits of the source datasets.

Stage 1: Instruction-to-Code Mapping. Off-the-shelf models lack familiarity with COM interfaces, frequently producing syntax errors or hallucinating non-existent API calls. To bootstrap a base policy π_θ, we construct an instruction–code corpus $\mathcal{D}_{\mathrm{S1}} = \{(\mathcal{I}_i, a_i)\}_{i=1}^{N}$, where frontier LLMs generate candidate COM programs for the training instructions and only executable pairs are retained. We fine-tune π_θ on $\mathcal{D}_{\mathrm{S1}}$ with the standard autoregressive objective. This stage equips the agent with the basic capability to generate executable COM programs, but the resulting single-turn policy cannot recover from execution failures.

Stage 2: Agentic Refinement with Multimodal Feedback. We next transform the single-turn policy into an interactive agent that can recover from failures. We roll out the Stage-1 policy in ComForge on a disjoint subset of the training data, collecting multi-turn trajectories containing screenshots, execution outputs, generated programs, and subsequent revisions. Trajectories that successfully complete the task are retained as positive supervision; for failed trajectories, a strong teacher model diagnoses the failure from the interaction history and rewrites the erroneous reasoning and program into a corrected trajectory. The resulting corpus $\mathcal{D}_{\mathrm{S2}}$ covers error diagnosis, iterative program revision, and task termination. We optimize π_θ with the following objective:

$$\mathcal{L}_{\mathrm{S2}}(\theta) = -\mathbb{E}_{\tau\sim\mathcal{D}_{\mathrm{S2}}}\left[\sum_{t=0}^{|\tau|}\sum_{k=1}^{|\mathcal{Z}_t|}\log \pi_\theta\Big(\mathcal{Z}_t^{(k)} \mid \mathcal{Z}_t^{(<k)}, o_{\le t}, \mathcal{Z}_{<t}, \mathcal{I}\Big)\right] \tag{1}$$

Table 2: Experimental results across single-task and multi-task settings. Each cell reports Code Valid Rate / Task Success Rate. Abbreviations: 3D-M = 3D model, Asm. = assembly, 2D-S = 2D sketch, E = edit, D = drawing, MP = mass properties, ID = interference detection, SW = SolidWorks, Inv. = Inventor, and ACAD = AutoCAD. Our agent is evaluated without few-shot prompting or retrieval augmentation.

Tasks VR / SR ↑	Single-Task (400)				Multi-Task (600)					
	3D-M (200)		Asm. (100)	2D-S (100)	3D-M+E (200)		3D-M+D (50)	3D-M+MP (200)		Asm.+ID
	SW (100)	Inv. (100)	Inv. (100)	ACAD (100)	SW (100)	Inv. (100)	SW (100)	SW (100)	Inv. (100)	Inv. (100)
GUI as Action										
GPT-5	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
Qwen3.5-9B	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
COM as Action										
GPT-5	24.0 / 7.0	28.0 / 21.0	5.0 / 2.0	71.0 / 33.0	24.0 / 7.0	19.0 / 15.0	0.0 / 0.0	13.0 / 6.0	25.0 / 10.0	0.0 / 0.0
Claude-Sonnet-4.6	7.0 / 5.0	8.0 / 8.0	3.0 / 1.0	63.0 / 52.0	4.0 / 2.0	4.0 / 4.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
GLM-4.6v	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
Qwen3.5-35B	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
Qwen3.5-9B	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
COM as Action with Retrieval Augmentation										
GPT-5	30.0 / 6.0	50.0 / 36.0	70.0 / 52.0	49.0 / 40.0	18.0 / 2.0	45.0 / 31.0	0.0 / 0.0	14.0 / 2.0	48.0 / 28.0	65.0 / 43.0
Claude-Sonnet-4.6	10.0 / 4.0	68.0 / 58.0	49.0 / 40.0	25.0 / 25.0	11.0 / 8.0	36.0 / 30.0	0.0 / 0.0	0.0 / 0.0	40.0 / 25.0	9.0 / 7.0
GLM-4.6v	0.0 / 0.0	6.0 / 4.0	10.0 / 8.0	0.0 / 0.0	0.0 / 0.0	4.0 / 4.0	0.0 / 0.0	0.0 / 0.0	2.0 / 2.0	0.0 / 0.0
Qwen3.5-35B	5.0 / 3.0	1.0 / 0.0	0.0 / 0.0	21.0 / 19.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	5.0 / 0.0	0.0 / 0.0	0.0 / 0.0
Qwen3.5-9B	0.0 / 0.0	1.0 / 1.0	4.0 / 2.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	2.0 / 0.0
COM as Action with Few-Shot Prompting										
GPT-5	64.0 / 36.0	69.0 / 60.0	67.0 / 56.0	98.0 / 58.0	66.0 /40.0	54.0 /34.0	50.0 / 43.0	58.0 / 40.0	62.0 / 26.0	62.0 / 42.0
Claude-Sonnet-4.6	76.0 / 74.0	82.0 / 60.0	57.0 / 55.0	96.0 / **88.0**	81.0 / **77.0**	56.0 / 40.0	69.0 / 22.0	73.0 / 51.0	72.0 / 34.0	52.0 / 34.0
GLM-4.6v	40.0 / 8.0	21.0 / 12.0	11.0 / 11.0	**100.0** / 82.0	40.0 / 25.0	12.0 / 8.0	34.0 / 2.0	30.0 / 4.0	4.0 / 4.0	0.0 / 0.0
Qwen3.5-35B	11.0 / 8.0	10.0 / 7.0	7.0 / 6.0	79.0 / 64.0	13.0 / 11.0	10.0 / 6.0	4.0 / 0.0	7.0 / 2.0	12.0 / 2.0	16.0 / 12.0
Qwen3.5-9B	17.0 / 10.0	18.0 / 11.0	18.0 / 18.0	92.0 / 72.0	18.0 / 13.0	10.0 / 4.0	6.0 / 0.0	8.0 / 0.0	8.0 / 4.0	16.0 / 16.0
Ours	**90.0 / 81.0**	**96.0 / 89.0**	**95.0 / 88.0**	86.0 / 86.0	**84.0** / 75.0	**97.0 / 87.0**	**76.0 / 61.0**	**78.0 / 58.0**	**98.0 / 86.0**	**95.0 / 80.0**

where o_t denotes the environment feedback and $\mathcal{Z}_t = (r_t, \delta_t, a_t)$ consists of the reasoning trace r_t, decision token δ_t, and predicted COM program a_t. After this stage, the agent can interpret environment feedback and iteratively debug its actions. However, a subtler failure mode remains: a program may execute without errors while still producing an artifact that violates the task specification.

Stage 3: Task-Level Optimization with Continuous Geometric Reward. To close the gap between code executability and task correctness, we further optimize the agent with on-policy RL using GRPO (Shao et al., 2024). We curate $\mathcal{D}_{\mathrm{S3}}$ from Stage-2 rollouts, retaining tasks whose programs execute successfully but fail task verification. For each instruction, the policy samples K candidate programs, which ComForge executes and scores by geometric fidelity computed via Chamfer distance (CD) between normalized point clouds P and Q sampled from the generated artifact and the ground truth (mesh surfaces for 3D STL, parametric curves for 2D DXF):

$$\mathrm{CD}(P, Q) = \frac{1}{|P|} \sum_{x \in P} \min_{y \in Q} \|x - y\|_2^2 + \frac{1}{|Q|} \sum_{y \in Q} \min_{x \in P} \|y - x\|_2^2. \tag{2}$$

To avoid reward sparsity, we convert CD into a continuous reward via log-linear decay:

$$R_{\mathrm{geo}}(\mathrm{CD}) = \mathbb{I}(\mathrm{CD} \le c_{\mathrm{low}}) \quad + \mathbb{I}(c_{\mathrm{low}} < \mathrm{CD} \le c_{\mathrm{high}}) \cdot \left(1 + \frac{\log \mathrm{CD} - \log c_{\mathrm{low}}}{\log c_{\mathrm{high}} - \log c_{\mathrm{low}}} \cdot (r_{\min} - 1)\right) \tag{3}$$

where $c_{\mathrm{low}}, c_{\mathrm{high}}$ are the near-perfect and failure thresholds, and $r_{\min} = 0.01$. Execution failures receive $R_{\mathrm{geo}} = 0$. GRPO normalizes these rewards within each group into relative advantages and updates π_θ with a clipped objective and a KL penalty toward the Stage-2 policy. Together, the three stages transform a single-turn COM code generator into a self-correcting agent capable of recovering from execution failures and improving task-level artifact fidelity.

7 Experiments

7.1 Experimental Setup

Benchmarks and Evaluation Metrics. Our primary benchmark is ComCADBench (Section 4). To test whether ComAct generalizes beyond CAD, we further evaluate off-the-shelf models on

Table 3: Zero-shot task success rate (%) on the COM-compatible subsets of WindowsAgentArena and ProSoftArena under GUI interaction and ComAct. L1/L2 denote ProSoftArena capability levels. GUI results are taken from the original benchmark papers.

(a) WindowsAgentArena

Model	OS	Office
Human	83.3	75.3
GUI		
Phi3-V	8.3	0.0
GPT-4o-mini	8.3	0.0
GPT-4o	29.2	2.3
GPT-4V	33.3	2.3
ComAct		
Qwen3.5-9B	33.3	26.3
Qwen2.5-VL-72B	36.4	26.3
GPT-5	**66.7**	**42.1**

(b) ProSoftArena

	Photoshop		Illustrator		Excel	
Model	L1	L2	L1	L2	L1	L2
GUI						
GLM-4.5v	0.0	0.0	0.0	0.0	0.0	0.0
GPT-4o	0.0	0.0	5.3	0.0	20.0	0.0
o3	37.5	7.7	31.6	0.0	35.0	8.3
Qwen2.5-VL-72B	16.7	0.0	5.3	0.0	35.0	8.3
GPT-5	40.0	7.7	47.4	6.7	65.0	8.3
ComAct						
Qwen3.5-9B	10.0	0.0	0.0	0.0	55.0	8.3
Qwen2.5-VL-72B	25.0	0.0	15.8	0.0	**75.0**	16.7
GPT-5	**45.0**	**30.7**	**78.9**	**20.0**	**75.0**	**50.0**

the COM-compatible Office, Adobe, and OS-shell tasks of two desktop agent benchmarks, WindowsAgentArena (Bonatti et al., 2024) and ProSoftArena (Ai et al., 2026). For CAD generation, we use the test sets of Text2CAD (Khan et al., 2024) (8,042 tasks) and the out-of-domain CADPrompt (Govindarajan et al., 2025) (200 tasks). On ComCADBench, we report the *Valid Rate* (VR), the fraction of tasks whose code executes without error, and the *Success Rate* (SR), the fraction whose final artifacts meet the task requirements. A 3D model or assembly counts as successful if its Chamfer distance to the ground truth is below 10^{-3}, and a 2D sketch if it is below ϵ_{2D}; an engineering drawing must contain all required views and annotations, and mass-property and interference results must match the ground truth element-wise. A multi-task pipeline succeeds only when every sub-task does. On ProSoftArena and WindowsAgentArena, we report task success using each benchmark's official evaluator. On Text2CAD and CADPrompt, we report the mean and median CD between generated and ground-truth shapes, together with VR.

Baselines and Implementation Details. We compare against two categories of baselines. GUI agents (GPT-5 (Singh et al., 2025) and Qwen3.5-9B (Qwen Team, 2026)) act through mouse and keyboard for up to 35 turns; COM agents (GPT-5, Claude-Sonnet-4.6 (Anthropic, 2026), GLM-4.6v (Zeng et al., 2025), and Qwen3.5-35B/9B) operate through COM programs under zero-shot, few-shot, or retrieval-augmented (RAG) prompting, sharing the same environment, prompt, and budget of five code revisions as our agent. On the external CAD benchmarks, we additionally compare with dedicated CAD generation models: Text2CAD, CAD-Coder (Guan et al., 2026), CADmium (Govindarajan et al., 2025), and CAD-Judge (Zhou et al., 2026). Our agent is initialized from Qwen3.5-9B and trained with `ms-swift` on 8 A100 GPUs using ComForge. The three training stages use 12K, 30K, and 2,156 samples, respectively; Stage 3 focuses on geometry-bearing tasks including 3D modeling, assembly, and 2D sketching. Full baseline configurations and training details are provided in Appendix B.

7.2 Main results on ComCADBench

ComAct Substantially Improves over GUI Interaction. As shown in Table 2, current GUI agents achieve near-zero success on ComCADBench, whereas switching to ComAct immediately yields non-trivial performance without task-specific training. For example, zero-shot GPT-5 reaches 33.0% SR on 2D sketching, and few-shot prompting further improves both code validity and task success. This large gap indicates that replacing long sequences of visually grounded GUI actions with executable COM programs substantially reduces the interaction difficulty of professional CAD workflows. To rule out weak grounding or a short budget as the cause of failure, we also test an enhanced variant built on GPT-5.4 with Set-of-Marks prompts and a 100-step budget. It still achieves zero SR on single tasks, while consuming 41× more time and 11.6× more tokens than the 35-turn agent. Its trajectories do not nearly finish; they stall early in misdirected plans and repeated erroneous clicks.

Our Agent Achieves Strong Performance on ComCADBench. Our 9B agent outperforms proprietary models such as GPT-5 and Claude-Sonnet-4.6 in both single-task and multi-task settings. The advantage is particularly pronounced on long-horizon workflows: while untrained baselines degrade sharply as errors propagate across dependent stages, our agent maintains 74.5% SR on multi-task pipelines, showing that the progressive training framework improves not only COM code generation, but also the agent's ability to recover from execution failures and produce task-correct final artifacts.

7.3 Results on WindowsAgentArena and ProSoftArena

To evaluate the effectiveness of ComAct beyond CAD, we test off-the-shelf models in a zero-shot setting on WindowsAgentArena (Bonatti et al., 2024) and ProSoftArena (Ai et al., 2026), covering OS Shell, Microsoft Office, Adobe Photoshop, and Illustrator. As shown in Table 3, ComAct consistently improves over GUI interaction. GPT-5 reaches 66.7%/42.1% on OS Shell/Office, and gains up to 41.7 points on ProSoftArena; similar improvements are also observed for open-source models, showing that the benefits of ComAct extend beyond CAD to diverse professional applications.

Table 4: Results on Text2CAD and CADPrompt. CD values are multiplied by 10^3.

Method	Mean CD↓	Median CD↓	VR↑
(a) Text2CAD			
Claude-3.7-Sonnet	186.53	134.16	52.97
GPT-4o	133.52	45.91	7.00
DeepSeek-V3	186.69	107.57	48.04
Qwen2.5-72B	209.41	153.81	17.36
Qwen2.5-7B	202.35	169.86	1.17
Text2CAD	29.29	0.37	96.25
CAD-Coder	<u>6.54</u>	0.17	**98.55**
Ours-SolidWorks	7.63	<u>0.09</u>	89.14
Ours-Inventor	**1.20**	**0.07**	<u>96.75</u>
(b) CADPrompt			
Text2CAD	–	127.70	**98.43**
CADmium	–	116.75	93.72
CAD-Judge	152.40	42.69	<u>97.45</u>
Ours-SolidWorks	**24.15**	**6.07**	88.75
Ours-Inventor	<u>26.47</u>	<u>6.69</u>	92.50

7.4 Results on Text2CAD and CADPrompt

We further compare our agent with dedicated CAD generation models on two public CAD generation benchmarks: Text2CAD (Khan et al., 2024) and CADPrompt (Govindarajan et al., 2025) in Table 4. Our agent achieves the best geometric fidelity on both benchmarks. On Text2CAD, the Inventor backend reduces the mean/median CD from 6.54/0.17 (CAD-Coder) to 1.20/0.07; on CADPrompt, the SolidWorks backend reduces it from 152.40/42.69 (CAD-Judge) to 24.15/6.07. Our VR is relatively lower, likely because COM programs for commercial CAD software must satisfy stricter interfaces than lightweight geometry libraries, but the programs that do execute produce substantially more accurate geometry.

7.5 Ablation Study

We ablate the three training stages on the single-task subset in Table 5. Stage 1 provides the basic ability to write COM programs but leaves many of them non-executable, especially on SolidWorks (58.0% VR). Stage 2 substantially improves VR across all tasks, showing that multi-turn environment feedback enables effective recovery from execution failures. However, a clear gap between VR and SR remains, indicating that executable programs may still produce incorrect artifacts. Stage 3 narrows this gap and consistently raises SR by directly optimizing a task-level geometric reward.

Table 5: Ablation of training stages on single tasks (VR / SR, %).

	3D-M		Asm.
Stages	SW	Inv.	Inv.
Stage 1	58.0 / 52.0	89.0 / 79.0	73.0 / 66.0
+ Stage 2	88.0 / 76.0	95.0 / 87.0	93.0 / 82.0
+ Stage 3	**90.0 / 81.0**	**96.0 / 89.0**	**95.0 / 88.0**

7.6 Further Analysis

Qualitative Case Study. Figure 3 shows a representative trajectory of our agent. When the initial COM program fails, the agent interprets the execution feedback, identifies the error, and revises the program in subsequent turns until completing the task. This example illustrates how multi-turn

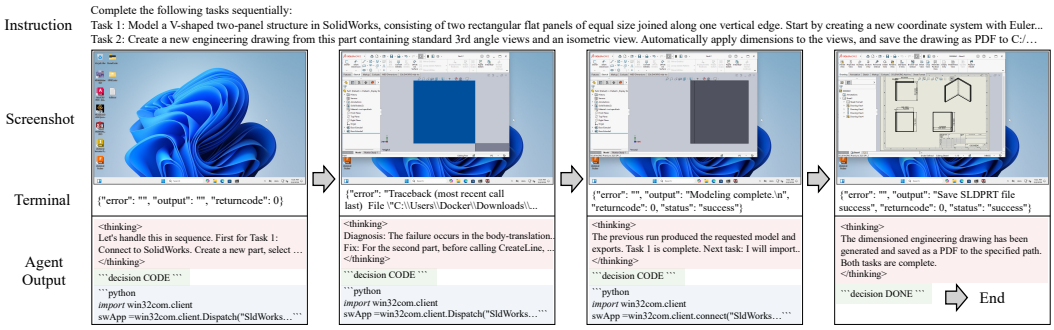

Figure 3: An execution trajectory of our agent completing a multi-task pipeline (3D modeling and engineering drawing).

interaction enables the agent to move beyond one-shot code generation and actively recover from execution failures.

Failure Analysis. We further categorize failures in Appendix C. Syntax errors dominate the failures of untrained models and concentrate on SolidWorks and Inventor, whose COM interfaces are not designed for Python: they require early binding, explicit interface casting, raw integer constants, and methods with over twenty positional arguments. Training largely removes these errors; for the same 9B backbone, syntax errors on Inventor drop from 42.9% to 5.1%. Many of the remaining failures of our agent are artifact mismatches and premature termination, which raise no execution error and must be detected from visual feedback alone.

RAG Underperforms Few-Shot Prompting. RAG lowers the average SR of GPT-5 from 43.5% under few-shot prompting to 24.0%. The bottleneck is not which API to call, but how to compose a runnable program under the conventions above, which API documentation does not describe but a single worked example conveys directly: on SolidWorks modeling, RAG raises GPT-5's VR only to 30.0%, whereas few-shot prompting lifts it to 64.0%. Long retrieved API lists, up to 49 per task, also displace task constraints from the context.

The Syntax–Geometry Gap. Few-shot prompting raises code validity, yet a large gap separates validity from success: averaged over ComCADBench, GPT-5 and Claude-Sonnet-4.6 reach 65.0% and 71.4% VR but only 43.5% and 53.5% SR, revealing a substantial syntax–geometry gap. These models imitate COM syntax well enough to avoid crashes, but lack the spatial reasoning to satisfy geometric constraints, producing programs that run yet build the wrong part.

8 Conclusion

We proposed ComAct, a unified COM-based action space for professional software agents that replaces long and fragile GUI interaction sequences with compact executable programs. To evaluate this paradigm in a demanding setting, we constructed ComCADBench, a benchmark of 1,000 tasks spanning three commercial CAD applications and multi-stage engineering workflows, and developed ComForge, a scalable platform for large-scale training and evaluation in real Windows software environments. Building on them, we trained a self-correcting 9B agent through progressive instruction-to-code learning, multi-turn refinement, and task-level reinforcement learning. Extensive experiments show that ComAct substantially improves the effectiveness and efficiency of professional software interaction over GUI-based agents, while the trained agent further narrows the gap between executable code and task-correct artifacts, particularly on long-horizon workflows. The paradigm also transfers beyond CAD to Office, Adobe, and OS-shell tasks, and the trained agent generalizes to the external CAD benchmarks Text2CAD and CADPrompt. Our approach is currently restricted to Windows applications, and errors that do not trigger execution failures remain challenging to detect. Extending training to more professional software and operating systems and strengthening visual verification are promising directions, and we hope the released benchmark, platform, and models will facilitate future research in these directions.

AI USE STATEMENT

Large language models are an object of study in this work: we evaluate them as baselines, use them to generate natural-language instructions and candidate COM programs for our benchmark and training data (all verified by execution, with manual review of a subset of the generated instructions), use a teacher model to correct failed trajectories, and use an LLM in the retrieval pipeline of the RAG baseline. In preparing the manuscript, we used LLMs to assist with writing, including drafting and revising text. The authors reviewed all content and take full responsibility for the paper.

REPRODUCIBILITY STATEMENT

The ComAct formulation is given in Section 3, benchmark construction in Section 4 and Appendix A, the ComForge environment in Section 5, and the training procedure in Section 6, with full hyperparameters in Appendix B.1 and baseline configurations and prompts in Appendix B. We will release ComForge, code, ComCADBench, training data, and model weights. Because commercial CAD software cannot be redistributed, ComForge will be released with non-activated software environments, and evaluation will be supported through a centralized submission platform.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

Jiaxin Ai, Yukang Feng, Fanrui Zhang, Jianwen Sun, Zizhen Li, Chuanhao Li, Yifan Chang, Wenxiao Wu, Ruoxi Wang, Mingliang Zhai, and Kaipeng Zhang. ProSoftArena: Benchmarking hierarchical capabilities of multi-modal agents in professional software environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 34586–34595, June 2026.

Anthropic. Developing a computer use model. `https://www.anthropic.com/news/developing-computer-use`, October 2024. Accessed: 2025-03-25.

Anthropic. Claude Sonnet 4.6 system card. Technical report, Anthropic, February 2026. URL `https://www.anthropic.com/claude/sonnet`.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows Agent Arena: Evaluating multi-modal OS agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. TransCAD: A hierarchical transformer for CAD sequence inference from point clouds. In *European Conference on Computer Vision*, pp. 19–36. Springer, 2024.

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. *Advances in Neural Information Processing Systems*, 38:59765–59789, 2026.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. CogAgent: A visual language model for GUI agents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14281–14290, 2024.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, et al. OS agents: A survey on MLLM-based agents for computer, phone and browser use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7436–7465, 2025.

Mohammad S Khan, Sankalp Sinha, Talha U Sheikh, Didier Stricker, Sk A Ali, and Muhammad Z Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 37:7552–7579, 2024.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. AutoWebGLM: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024.

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*, pp. 18563–18573, 2025a.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8778–8786, 2025b.

Microsoft. Component Object Model (COM). `https://learn.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal`, 2024. Accessed: 2025-05-26.

OpenAI. Gpt-4o system card. `https://cdn.openai.com/gpt-4o-system-card.pdf`, 2024. Accessed: 2024-09-26.

OpenAI. Computer-using agent. `https://openai.com/index/computer-using-agent/`, January 2025. Accessed: 2025-03-25.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen3.5. `https://qwenlm.github.io/blog/qwen3.5/`, February 2026. Accessed: 2026-05-26.

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

Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P Adams. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. *arXiv preprint arXiv:2007.08506*, 2020.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. OpenAI GPT-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, et al. CoAct-1: Computer-using agents with coding as actions. *arXiv preprint arXiv:2508.03923*, 2025a.

Yueqi Song, Frank F Xu, Shuyan Zhou, and Graham Neubig. Beyond browsing: API-based web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11066–11085, 2025b.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*, 2024.

Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 Gallery: A dataset and environment for programmatic CAD construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4):1–24, 2021.

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*, pp. 6772–6782, 2021.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-Copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. Aria-UI: Visual grounding for GUI instructions. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 22418–22433, 2025.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, et al. Large language model-brained GUI agents: A survey. *arXiv preprint arXiv:2411.18279*, 2024.

Chaoyun Zhang, Liqun Li, He Huang, Chiming Ni, Bo Qiao, Si Qin, Yu Kang, Minghua Ma, Qingwei Lin, Saravan Rajmohan, et al. UFO^3: Weaving the digital agent galaxy. *arXiv preprint arXiv:2511.11332*, 2025a.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. AppAgent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025b.

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 *International Conference on Learning Representations*, volume 2025, pp. 3204–3227, 2025c.

Yuze Zhao, Jintao Huang, Jinghan Hu, et al. SWIFT: A scalable lightweight infrastructure for fine-tuning. *arXiv preprint arXiv:2408.05517*, 2024.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024.

Zheyuan Zhou, Jiayi Han, Liang Du, Naiyu Fang, Lemiao Qiu, and Shuyou Zhang. CAD-Judge: Toward efficient morphological grading and verification for text-to-CAD generation. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1021–1025. IEEE, 2026.

Table 6: Overall statistics of **ComCADBench**.

Complexity	Task Category	Software	# Samples
Single-Task ($N = 400$)	3D Modeling	SolidWorks, Inventor	200
	Assembly	Inventor	100
	2D Draft / Sketch	AutoCAD	100
Multi-Task ($N = 600$)	3D Modeling + Edit	SolidWorks, Inventor	200
	3D Modeling + Drawing	SolidWorks, Inventor	100
	3D Modeling + Mass Prop.	SolidWorks, Inventor	200
	Assembly + Interference	Inventor	100
Total	**All 7 Categories**	**SW, Inv, ACAD**	**1,000**

Table 7: Statistics of the crawled COM API database across three CAD platforms. The filtered subset represents APIs directly relevant to the core engineering tasks evaluated in ComCADBench.

CAD Software	Total Crawled	Task-Related Filtered
SolidWorks	12,376 (Interfaces) + 961 (Const)	3,302 + 38
Inventor	24,269 (Objects) + 451 (Enums)	1,509 + 11
AutoCAD	4,078 (Drawing/File only)	4,078

Table 8: Number of retrieved APIs for tasks across task categories and software.

Task Category & Software	Retrieved APIs (Avg / Max / Min)
2D Sketch	
- AutoCAD	9.45 / 10 / 7
3D Model	
- SolidWorks	17.45 / 24 / 12
- Inventor	38.17 / 39 / 35
Assembly	
- Inventor	26.00 / 26 / 26
3D Model + Modify	
- SolidWorks	17.98 / 20 / 12
- Inventor	38.30 / 39 / 35
3D Model + Draw	
- SolidWorks	30.45 / 37 / 25
3D Model + Mass Property	
- SolidWorks	33.06 / 38 / 27
- Inventor	47.40 / 49 / 41
Assembly + Interference	
- Inventor	35.00 / 35 / 35

A Data Construction

To ensure the high quality and practical relevance of ComCADBench, we employ a rigorous, multi-stage data construction pipeline. This process transforms raw parametric data into natural language instructions paired with executable COM scripts, verified through physical execution in our ComForge environment.

A.1 Source Data and Textualization

We construct the foundation of our benchmark by sourcing high-quality parametric CAD designs from three public datasets: Text2CAD (Khan et al., 2024) (for 3D modeling), SketchGraphs (Seff

et al., 2020) (for 2D sketching), and Fusion360Gallery (Willis et al., 2021) (for assembly). These platforms natively provide structured parametric descriptions of models in JSON format, which contain all necessary geometric entities and topological constraints.

Since SketchGraphs and Fusion360Gallery do not provide natural language instructions, we develop an automated textualization pipeline. We prompt a frontier MLLM (GPT-5 (Singh et al., 2025)) to ingest the structured JSON files and generate corresponding professional engineering instructions. To guarantee linguistic diversity and professional accuracy, a subset of the generated text instructions undergoes rigorous human evaluation. For Text2CAD, we directly utilize its high-quality human-annotated text instructions.

A.2 Ground Truth COM Script Synthesis

With the structured JSON files containing complete topological and geometric information, we employ a rule-based engine to synthesize the Ground Truth (GT) COM scripts. For instance, in 2D sketch tasks, our parser extracts entity types (e.g., lines, arcs, circles) and their precise coordinates from the JSON, subsequently mapping them to native COM APIs to generate the deterministic executable script. To illustrate this mapping, Table 9 provides a side-by-side comparison of the raw JSON representation and the synthesized COM script for a standard sketching operation.

A.3 Downstream Multi-Task Construction

Beyond foundational modeling, real engineering workflows require diverse downstream operations. We construct the remaining task categories as follows:

- **Mass Properties Analysis:** We predefine a comprehensive list of analytical properties, including mass, volume, density, center of mass, principal moments, principal rotation (for Inventor), principal axes (for SolidWorks), inertia tensor at centroid, and inertia tensor at origin. We randomly pair these properties with 8 standard industrial materials (Water, Carbon Steel, Alloy Steel, Stainless Steel 304, Aluminum 6061, Brass, ABS Plastic, Nylon 6/6) to generate instructions and construct the corresponding GT scripts.
- **Interference Detection:** We implement two levels of collision analysis: Hard Interference, which checks for physical volume overlaps between intersecting components, and Soft Interference, which evaluates clearance boundaries and bounding box intersections. GT scripts are generated to sequentially invoke the respective CAD detection APIs.
- **Model Editing:** We define a set of fundamental modification operations: adding/deleting entities, modifying dimensions, applying fillets/chamfers, and drilling holes. We prompt GPT-5 to select from these operations, formulate an edit instruction, and modify the original modeling COM script accordingly.

Cross-Model Consensus for Edit Verification: Since LLM-generated edit scripts cannot be guaranteed correct by rule-based generation alone, we introduce a dual-model consensus mechanism. Using the same edit instruction and original script, we prompt a secondary model (Qwen-32B) to generate an independent edit script. Both scripts are executed, and if the Chamfer Distance (CD) between their resulting 3D artifacts is below a strict threshold ($\epsilon \leq 10^{-4}$), the sample is deemed valid. We retain the edit instruction, the GT code, and the output artifact. A random subset is additionally verified by human experts to ensure dataset reliability.

Multi-Task Pipelines: Long-horizon tasks are constructed by composing single-turn instructions sequentially (e.g., executing a 3D assembly followed immediately by interference detection).

A.4 Execution Verification in ComForge

To guarantee strict executability, all synthesized GT scripts across all task categories are deployed and executed within the Dockerized Windows environments of ComForge. The artifacts generated by the scripts are extracted and compared against the original dataset's ground truth models. Only samples whose script-generated artifacts achieve a Chamfer Distance below the strict threshold against the true GT are retained in the final ComCADBench dataset.

Table 9: An example of mapping raw parametric JSON data to the corresponding COM script.

Parametric JSON Representation


```
{
  "sketch_id": "f77bdbb84580a54d22076c0e_0_0",
  "entities": [
    {
      "type": "Line",
      "entityId": "1",
      "isConstruction": false,
      "dirX": 1.0,
      "dirY": 0.0,
      "pntX": 0.0014664754271507263,
      "pntY": 0.012719333171844482,
      "startParam": -0.07332348078489304,
      "endParam": 0.07332348078489304
    },
    {
      "type": "Line",
      "entityId": "4",
      "isConstruction": false,
      "dirX": 1.0,
      "dirY": 0.0,
      "pntX": 0.0014664754271507263,
      "pntY": 0.0,
      "startParam": -0.07332348078489304,
      "endParam": 0.07332348078489304
    },
    {
      "type": "Line",
      "entityId": "7",
      "isConstruction": false,
      "dirX": 0.0,
      "dirY": -1.0,
      "pntX": -0.07185700535774231,
      "pntY": 0.006359666585922241,
      "startParam": -0.006359666585922241,
      "endParam": 0.006359666585922241
    },
    {
      "type": "Line",
      "entityId": "10",
      "isConstruction": false,
      "dirX": 0.0,
      "dirY": -1.0,
      "pntX": 0.07478995621204376,
      "pntY": 0.006359666585922241,
      "startParam": -0.006359666585922241,
      "endParam": 0.006359666585922241
    },
    {
      "type": "Arc",
      "entityId": "13",
      "isConstruction": false,
      "xCenter": -0.0718570053577423,
      "yCenter": 0.006359666585922241,
      "xDir": 1.0,
      "yDir": 0.0,
      "radius": 0.006359666585922241,
      "clockwise": false,
      "startParam": -4.712388980384688,
      "endParam": -1.5707963267948988
    }
  ]
}
```


Synthesized GT COM Script (Python)

```
# --- Sketch ---
# Entity 1
points_1 = get_line_points(
    pnt_x = 0.0014664754271507263,
    pnt_y = 0.012719333171844482,
    dir_x = 1.0,
    dir_y = 0.0,
    start_param = -0.07332348078489304,
    end_param = 0.07332348078489304,
)
start_pnt_1 = win32com.client.VARIANT((
    points_1['start_x']*M_to_MM,
    points_1['start_y']*M_to_MM,
    0.0
))
end_pnt_1 = win32com.client.VARIANT((
    points_1['end_x']*M_to_MM,
    points_1['end_y']*M_to_MM,
    0.0
))
model_space.AddLine(start_pnt_1, end_pnt_1)

# Entity 4
points_4 = get_line_points(
    pnt_x = 0.0014664754271507263,
    pnt_y = 0.0,
    dir_x = 1.0,
    dir_y = 0.0,
    start_param = -0.07332348078489304,
    end_param = 0.07332348078489304,
)
start_pnt_4 = win32com.client.VARIANT((
    points_4['start_x']*M_to_MM,
    points_4['start_y']*M_to_MM,
    0.0
))
end_pnt_4 = win32com.client.VARIANT((
    points_4['end_x']*M_to_MM,
    points_4['end_y']*M_to_MM,
    0.0
))
model_space.AddLine(start_pnt_4, end_pnt_4)

# Entity 7
points_7 = get_line_points(
    pnt_x = -0.07185700535774231,
    pnt_y = 0.006359666585922241,
    dir_x = 0.0,
    dir_y = -1.0,
    start_param = -0.006359666585922241,
    end_param = 0.006359666585922241,
)
start_pnt_7 = win32com.client.VARIANT((
    points_7['start_x']*M_to_MM,
    points_7['start_y']*M_to_MM,
    0.0
))
end_pnt_7 = win32com.client.VARIANT((
    points_7['end_x']*M_to_MM,
    points_7['end_y']*M_to_MM,
    0.0
))
model_space.AddLine(start_pnt_7, end_pnt_7)
...
```

A.5 Data Visualization and Statistics

The detailed statistical distribution of ComCADBench across different task types and software platforms is presented in Table 6. We visualize GT artifacts in representative samples for 3D modeling

(Figure 4), assembly (Figure 6), and 2D sketching (Figure 5) tasks. Figure 7 provides instruction examples for each specific task category.

A.6 Detailed Empirical Insights into CAD Paradigms

To better understand the mechanisms driving CAD automation, we analyze the performance of various baseline models under different interaction paradigms and prompt configurations. This analysis reveals several critical insights:

Limitations of GUI-Based Paradigms. GUI-based paradigms fundamentally fail on professional CAD tasks. As evidenced by the top rows of Table 2, both GPT-5 and Qwen3.5-9B achieve a 0.0% SR and VR across all categories. Industrial CAD interfaces feature dense micro-icons and complex 3D viewports where MLLMs struggle with precise pixel-level grounding. Furthermore, continuous spatial reasoning errors accumulate inexorably across long-horizon trajectories, causing the execution to collapse before task completion.

Unlocking Full Reasoning Capability via the COM Paradigm. By shifting the interaction from fragile visual perception to deterministic code synthesis, the COM paradigm instantly unlocks non-trivial performance for proprietary models without domain-specific training. Table 2 clearly highlights this step-function improvement: in a zero-shot setting, GPT-5 and Claude-Sonnet-4.6 immediately jump from complete failure to meaningful task completion, with GPT-5 achieving up to 71.0% VR and 33.0% SR on 2D sketches. This validates that generalist LLMs can project their inherent coding proficiency into professional software via COM.

Task Complexity and Scale Discrepancies. Tasks with well-structured, flat API surfaces benefit most from the COM paradigm. 2D sketching in AutoCAD, for instance, maps cleanly onto a linear sequence of coordinate-based API calls, enabling GPT-5 to reach 58% SR with few-shot prompting. 3D modeling, in contrast, demands multi-coordinate-system reasoning, feature tree manipulation, and geometric constraint solving, which remains challenging even for strong models. Furthermore, a pronounced gap separates proprietary models from open-source alternatives. Qwen3.5-9B and Qwen3.5-35B barely exceed zero on most tasks without in-context support, indicating that effective COM manipulation relies heavily on pre-training exposure to domain-specific APIs. This gap widens further on multi-task pipelines, where errors in early sub-tasks propagate across stages, causing all untrained baselines to degrade sharply.

The Syntax-Geometry Gap in Advanced Prompting. While few-shot prompting significantly elevates the code valid rate, a massive discrepancy remains between code validity and geometric accuracy for generalist LLMs. This reveals a fundamental limitation: models can successfully mimic COM syntax to avoid execution crashes, yet lack the spatial reasoning required to satisfy engineering constraints. They frequently produce code that runs perfectly but yields geometrically incorrect results. This gap highlights the necessity of our RL alignment, which explicitly targets geometric intent rather than syntactic correctness.

Counterproductive Effects of Retrieval-Augmented Generation. On several tasks, RAG even degrades performance relative to zero-shot prompting. This is particularly noticeable for GPT-5, where additional API descriptions appear to disrupt rather than complement the model's existing COM knowledge. We attribute this to two factors: first, the retrieval corpus cannot distinguish between common and rare API usage patterns, occasionally surfacing functionally correct but idiomatically awkward APIs that lead to syntactically valid yet geometrically nonsensical code. Second, the RAG prompt occupies a substantial portion of the context window, displacing critical information about task constraints and environment states. This negative effect is most pronounced on multi-task pipelines, where coordinating across sub-tasks already strains the model's context utilization.

B Implementation Details

B.1 Training Implementation and Hyperparameters

All training stages of our agent are conducted on 8 NVIDIA A100 (80GB) GPUs. Our model is initialized from the Qwen3.5-9B (Qwen Team, 2026) backbone. The training pipeline is imple-

mented using the ms-swift framework, which provides robust support for both parameter-efficient fine-tuning and large-scale reinforcement learning.

Stage 1 & Stage 2 (Supervised Fine-Tuning): For both single-turn (Stage 1) and multi-turn (Stage 2) SFT, we apply Low-Rank Adaptation (LoRA) to all linear layers with a rank of $r = 8$ and $\alpha = 32$. The models are trained using the AdamW optimizer with a learning rate of 1e-5, a cosine learning rate scheduler, and a warmup ratio of 0.05. To accommodate the extensive context required for code generation and error tracebacks, the maximum sequence length is set to 16,384 tokens. DeepSpeed ZeRO-2 is enabled to optimize memory consumption.

Stage 3 (Group Relative Policy Optimization - GRPO): For the RL stage, we transition to full-parameter fine-tuning to maximize the model's capacity for geometric alignment. The training utilizes 2,156 curated samples focusing on 3D modeling, assembly, and 2D sketching. We integrate vLLM directly into the ms-swift pipeline (Colocate mode) to accelerate policy rollout generation. For each prompt, the model samples 4 distinct completions (generations) with a temperature of 1.0. To evaluate the rewards, we deploy 64 concurrent Dockerized Windows VMs within our ComForge infrastructure, providing synchronous real-time feedback. DeepSpeed ZeRO-3 is employed for distributed training.

B.2 Baseline Configurations and Prompting Strategies

To systematically evaluate the capability of generalist LLMs on CAD manipulation, we test baselines under Zero-shot, Few-shot, and Retrieval-Augmented Generation (RAG) settings. The system prompt universally assigns the model a professional role, specifies the target software, and strictly constrains the output format to executable Python COM scripts enclosed in markdown code blocks. Detailed prompts under each setting are provided in Appendix D.

Few-Shot Prompting. For complex spatial tasks such as 3D modeling and multi-task pipelines (assembly followed by interference detection), pure zero-shot generation often struggles with precise coordinate transformations and topological constraints. In the few-shot setting, we prepend one high-quality demonstration exemplar to the prompt context of each task category. These exemplars are manually curated to demonstrate the standard API workflow and correct spatial reasoning patterns for the target software.

Retrieval-Augmented Generation. To equip generalist MLLMs with the necessary domain knowledge for COM programming, we implement a Retrieval-Augmented Generation (RAG) pipeline that crawls, filters, and retrieves precise API endpoints to construct the prompt context. First, we crawled the official COM API documentation for SolidWorks, Inventor, and AutoCAD. To minimize database noise, we filtered the APIs to retain those directly relevant to our evaluated tasks. For AutoCAD, we restricted the crawl to drawing and file-related collections. The statistics of the constructed API database are detailed in Table 7, the retrieved API breakdowns are listed in Table 8, the standardized JSON schemas used to represent the APIs for each software are illustrated in Table 10. Next, we prompt GPT-4o (OpenAI, 2024) to extract a set of technical sub-queries from the natural language instruction. These sub-queries are then embedded using OpenAI's text-embedding-3-large model and searched against our vectorized API database to retrieve a preliminary pool of candidate APIs. Finally, we employ GPT-4o as a reranker. The reranker evaluates the candidate pool against the original sub-queries and selects the top 5 most relevant APIs per query. These finalized APIs—complete with their descriptions, parameters, and return types—are then injected into the prompt of the target LLM. The quantitative retrieval performance across different task categories is summarized in Table 8.

C Failure Analysis

Table 11 breaks down task failures into code-level and decision-level categories. Code-level failures include syntax errors, reported separately for each CAD application, and artifact mismatches, where the code executes cleanly but the resulting geometry, drawing, or analysis properties do not match the ground truth. Decision-level failures include *premature DONE*, where the agent declares completion while the task is unfinished, and *self-declared FAIL*, where the agent gives up before exhausting its budget. GPT-5 and Qwen3.5-9B are evaluated with few-shot prompting; ours is the trained 9B agent.

Table 10: Examples of the standardized JSON schemas used to represent crawled COM APIs in the RAG database. Differences in dictionary keys (e.g., `owner_interface` vs. `parent_Object`) naturally reflect the underlying COM architectures of the respective software.

Software: **AutoCAD**

```
{
    "type": "method",
    "parent_Object": "Document",
    "name": "SaveAs",
    "description": "Saves the document to a specified file.",
    "parameters": [],
    "return_value": "No return value."
}
```

Software: **SolidWorks**

```
{
    "type": "method",
    "owner_interface": "ISldWorks",
    "name": "NewDocument",
    "description": "Creates a new document based on the specified template.",
    "parameters": [
        {
            "name": "TemplateName",
            "description": "Fully qualified path and name of the template..."
        },
        {
            "name": "PaperSize",
            "description": "Size of paper as defined in swDwgPaperSizes_e"
        }
    ],
    "return_value": "Newly created document or NULL if the operation fails"
}
```

Software: **Inventor**

```
{
    "type": "method",
    "parent_Object": "FileManager",
    "name": "GetTemplateFile",
    "description": "Method that specifies the template to use when creating a file.",
    "parameters": [
        {
            "Name": "DocumentType",
            "Type": "DocumentTypeEnum",
            "Description": "Input constant that specifies the type of to create."
        },
        {
            "Name": "SystemOfMeasure",
            "Type": "SystemOfMeasureEnum",
            "Description": "Input constant that specifies the system of measure..."
        }
    ]
}
```

Syntax errors dominate the failures of untrained models and concentrate on SolidWorks and Inventor. Our training substantially reduces them; for the same 9B backbone, syntax errors on Inventor drop from 42.9% to 5.1%. Artifact mismatches are harder to self-correct: since the code executes cleanly, the agent receives no explicit error signal and must rely on screenshots alone to judge task completion.

Table 11: Per-category failure breakdown (%). GPT-5 and Qwen3.5-9B use few-shot prompting; ours is the trained 9B agent.

		Syntax error			Artifact mismatch			Decision error	
Model	SR	SW	Inv.	ACAD	Geometry	Drawing	Properties	Premature DONE	Self-declared FAIL
GPT-5	43.5	29.7	27.3	1.5	16.1	5.1	19.8	2.9	0.4
Qwen3.5-9B	14.8	43.8	42.9	4.0	3.2	3.0	3.0	1.2	0.0
Ours (9B)	79.1	24.0	5.1	18.7	10.8	20.0	21.3	8.7	0.0

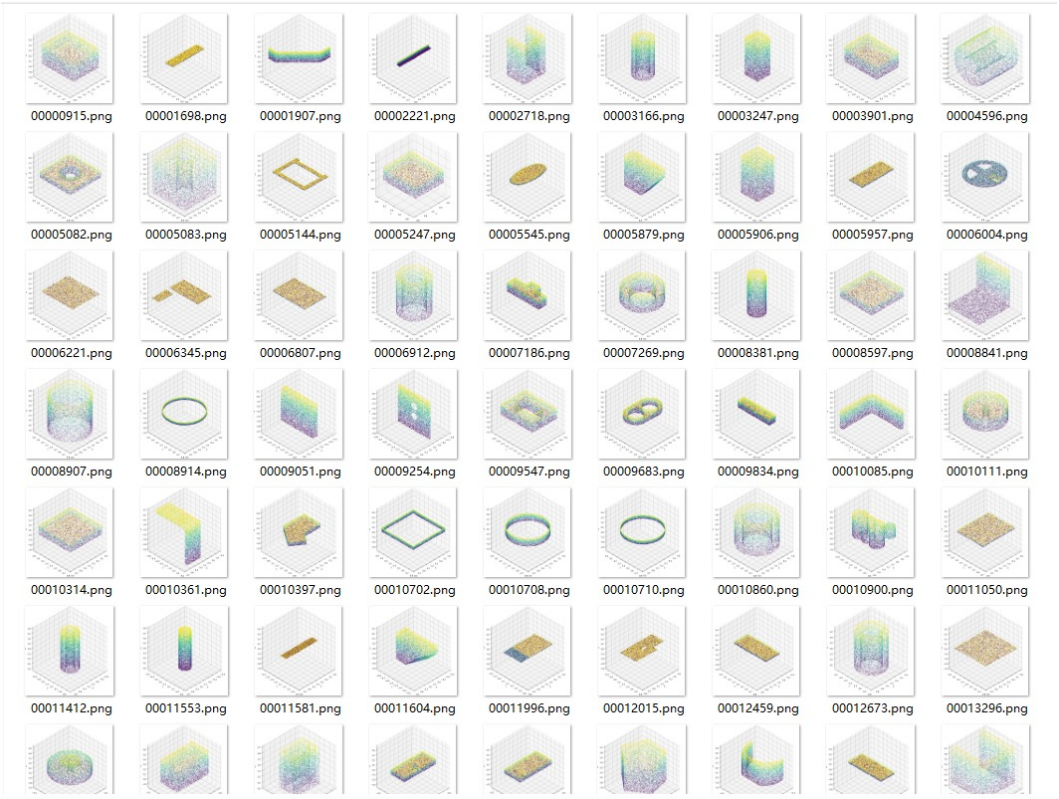

Figure 4: Visualization of the ground truth artifacts for 3d modeling samples in ComCADBench.

D PROMPTS

To evaluate the model's capabilities across different scenarios, we constructed three distinct system prompts: Baseline, Few-Shot, and Retrieval-Augmented Generation (RAG). All prompts share a foundational structure that defines the persona, execution rules, and input/output formats, but they differ in the provided context. The prompts used for our training and evaluation are shown in Table 12.

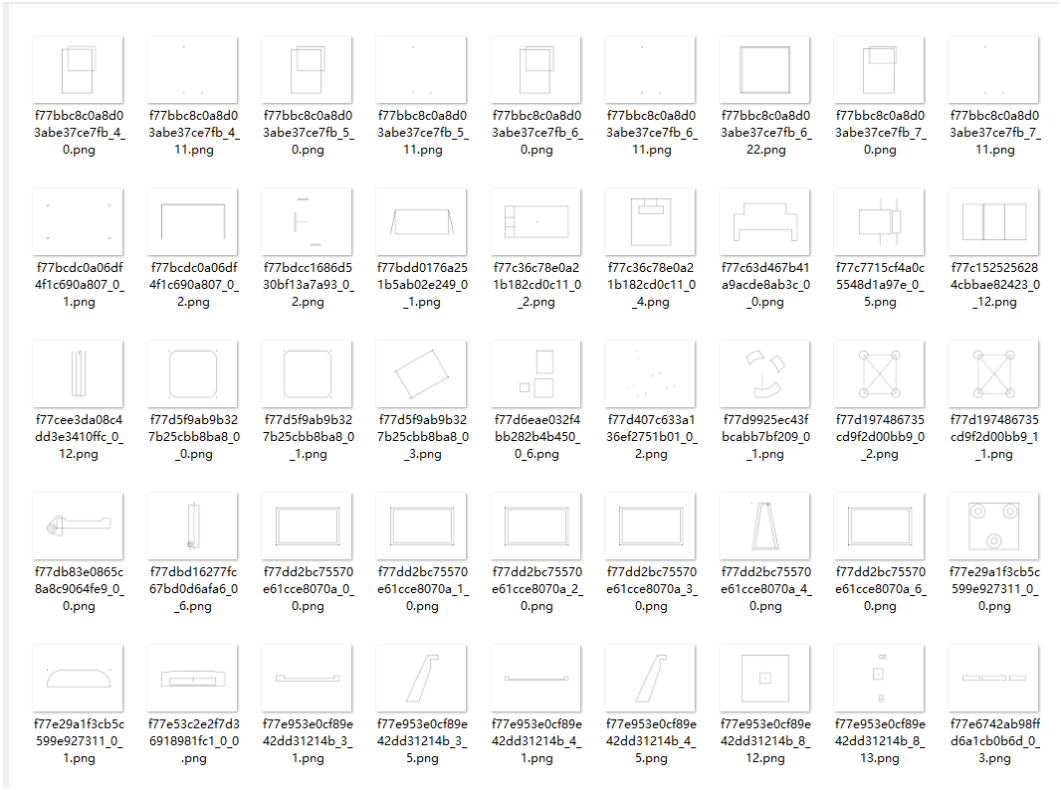

Figure 5: Visualization of the ground truth artifacts for 2d sketching samples in ComCADBench.



Figure 6: Visualization of the ground truth artifacts for assembly samples in ComCADBench.

3D Modeling

Create a new coordinate system by setting the Euler angles to [0.0, 0.0, -90.0] and the translation vector to [0.0, 0.75, 0.0]. Next, draw a two-dimensional sketch on a new face. This sketch should consist of a closed loop made up of four lines: the first line starts at (0.0, 0.0) and ends at (0.6889, 0.0); the second line starts at (0.6889, 0.0) and ends at (0.6889, 0.5409); the third line starts at (0.6889, 0.5409) and ends at (0.0, 0.5409); and the fourth line starts at (0.0, 0.5409) and ends at (0.0, 0.0). Transform the two-dimensional sketch into a three-dimensional object by rotating it using the Euler angles [0.0, 0.0, -90.0] and translating it using the translation vector [0.0, 0.75, 0.0]. Extrude the transformed two-dimensional sketch along the normal direction by 0.375 units, ensuring that no extrusion occurs in the opposite direction. Finally, create a new solid body from the extruded shape. The final dimensions of the rectangular prism are length 0.6888698326771653 units, width 0.5409246555111811 units, and height 0.375 units. After creating the solid body, export the model as an STL file to C:/Users/Docker/Downloads/00000915.stl and as a STEP file to C:/Users/Docker/Downloads/00000915.step.

2D Sketching

Initialize a planar sketch on the XY datum comprising the following geometric primitives: Entity 1: Draw a line passing through the reference point (-0.0001012963970691675 4, 2.7142287790865752e-05) along the direction vector (0.9659258262890684, -0.2588190451025205), defined by a start parameter of -0.0303395407460296 77 and an end parameter of -0.0049395407460296 64. Entity 4: Draw a line passing through the reference point (-0.0174862222556375 65, 0.0194901054946 6597) along the direction vector (-1.0, -1.2246467991473532e-16), defined by a start parameter of 0.0088097972577855 2 and an end parameter of 0.0150958210477117 7. Entity 7: Draw a line passing through the reference point (-0.0325820423614087 4, 0.0080942440072394 58) along the direction vector (1.2246467991473532e-16, -1.0), defined by a start parameter of -0.0113958614874261 36 and an end parameter of 0.0099636545003162 02. Entity 10: Draw a line passing through the reference point (0.0054780465275202 37, -0.0018694104930767 452) along the direction vector (1.0, 0.0), defined by a start parameter of -0.0380600888892898 and an end parameter of -0.0023030465275202 367. After drafting the entities, export the DXF file to: C:\\Users\\Docker\\Downloads\\f7a0a2252_0_0.dxf. Note: All given coordinates and dimensions are in meters and must be converted to millimeters in AutoCAD.

Assembly

Commence the assembly by grounding component 51862_e5f65013_0010_1 (located at \"C:\\Users\\Docker\\Documents\\AssemblyJoint\\steps\\51862_e5f6513_0010_1.step\") as the Fixed Base Component and 51862_e5f65013_0011_2 (located at \"C:\\Users\\Docker\\Documents\\AssemblyJoint\\steps\\51862_e5f6513_0011_2.step\") as the Moving Component. Designate the planar B-Rep face on the base component (reference entity point: {'x': -0.35, 'y': 2.90, 'z': 4.10}) and the planar B-Rep face on the moving component (reference entity point: {'x': -0.05, 'y': 4.89, 'z': 9.89}) as the mating entities. Establish a single-degree-of-freedom Revolute Joint by aligning the Center of the planar B-Rep face on the base component and the Center of the planar B-Rep face on the moving component. The constraint is configured in an Anti-Aligned (Flipped) orientation to ensure opposing primary vectors. Set the axial offset to zero to achieve coincidence between the mating entities. This configuration locks all relative translational and tilting degrees of freedom (DOF), preserving only a single rotational DOF about the synchronized axis anchored at the designated key points. After fully constraining the assembly, export the model as an STL file to: C:\\Users\\Docker\\Downloads\\joint_set_00006.stl and as a STEP file to: C:\\Users\\Docker\\Downloads\\joint_set_00006.step.

3D Model + Engineering Drawing

Complete the following tasks sequentially:\nTask 1: Model this part in Solidworks: Create a new coordinate system by setting the Euler angles to [0.0, 0.0, -90.0] and the translation vector to [0.0, 0.75, 0.0]. Next, draw a two-dimensional sketch on a new face. This sketch should consist of a closed loop made up of four lines: the first line starts at (0.0, 0.0) and ends at (0.6889, 0.0); the second line starts at (0.6889, 0.0) and ends at (0.6889, 0.5409); the third line starts at (0.6889, 0.5409) and ends at (0.0, 0.5409); and the fourth line starts at (0.0, 0.5409) and ends at (0.0, 0.0). Transform the two-dimensional sketch into a three-dimensional object by rotating it using the Euler angles [0.0, 0.0, -90.0] and translating it using the translation vector [0.0, 0.75, 0.0]. Extrude the transformed two-dimensional sketch along the normal direction by 0.375 units, ensuring that no extrusion occurs in the opposite direction. Finally, create a new solid body from the extruded shape. The final dimensions of the rectangular prism are length 0.68 units, width 0.54 units, and height 0.375 units. **After creating the solid body, export the model as an STL file to C:/Users/Docker/Downloads/00000915.stl and as a STEP file to C:/Users/Docker/Downloads/00000915.step.**\nTask 2: Create a new engineering drawing from this part containing standard 3rd angle views and an isometric view. Automatically apply dimensions to the views, and save the drawing as a PDF to C:\\Users\\Docker\\Downloads\\00000915.pdf.

Assembly + Interference Detection

Complete the following tasks sequentially:\nTask 1: Build this assembly in Inventor: Commence the assembly by grounding component 50379_ebec8fae_0003_1 (located at \"C:\\Users\\Docker\\Documents\\AssemblyJoint\\steps\\50379_1.step\") as the Fixed Base Component and 50379_ebec8fae_0002_2 (located at \"C:\\Users\\Docker\\Documents\\AssemblyJoint\\steps\\50379_2.step\") as the Moving Component. Designate the cylindrical B-Rep face on the base component (reference entity point: {'x': -15.23, 'y': 3.33, 'z': 0.0}) and the 3D B-Rep circular edge on the moving component (reference entity point: {'x': 0.84, 'y': 0.0, 'z': 0.63}) as the mating entities. Establish a single-degree-of-freedom Revolute Joint by aligning the axial Start-Point of the cylindrical B-Rep face on the base component and the Center of the 3D B-Rep circular edge on the moving component. The constraint is configured in an Anti-Aligned (Flipped) orientation to ensure opposing primary vectors. Set the axial offset to zero to achieve coincidence between the mating entities. This configuration locks all relative translational and tilting degrees of freedom (DOF), preserving only a single rotational DOF about the synchronized axis anchored at the designated key points. After fully constraining the assembly, export the model as an STL file to: C:\\Users\\Docker\\Downloads\\joint_set_00186.stl and as a STEP file to: C:\\Users\\Docker\\Downloads\\joint_set_00186.step.\nTask 2: Based on the assembly obtained in Task 1, perform a full automated interference analysis across all components in Inventor. Count the total number of components (bottom-level parts) in the assembly and extract their names, along with the total interference count and combined interference volume. Do not perform any unit conversions on the extracted values. Format the result strictly as a JSON object: {\"components\": [{\"count\": <component_count>, \"names\": [\"<name1>\", \"<name2>\",...]}, \"interference\": [{\"count\": <interference_count>, \"total_volume\": <volume_value>}]}. Save this JSON file to \"C:/Users/Docker/Downloads/joint_set_00186.json\".

3D Model + Mass Properties Analysis

Complete the following tasks sequentially:\nTask 1: Model this part in Solidworks: Create a new coordinate system by setting the Euler angles to [0.0, 0.0, -90.0] and the translation vector to [0.0, 0.75, 0.0]. Next, draw a two-dimensional sketch on a new face. This sketch should consist of a closed loop made up of four lines: the first line starts at (0.0, 0.0) and ends at (0.6889, 0.0); the second line starts at (0.6889, 0.0) and ends at (0.6889, 0.5409); the third line starts at (0.6889, 0.5409) and ends at (0.0, 0.5409); and the fourth line starts at (0.0, 0.5409) and ends at (0.0, 0.0). Transform the two-dimensional sketch into a three-dimensional object by rotating it using the Euler angles [0.0, 0.0, -90.0] and translating it using the translation vector [0.0, 0.75, 0.0]. Extrude the transformed two-dimensional sketch along the normal direction by 0.375 units, ensuring that no extrusion occurs in the opposite direction. Finally, create a new solid body from the extruded shape. The final dimensions of the rectangular prism are length 0.6888698326771653 units, width 0.5409246555111811 units, and height 0.375 units. **After creating the solid body, export the model as an STL file to C:/Users/Docker/Downloads/00000915.stl and as a STEP file to C:/Users/Docker/Downloads/00000915.step.**\nTask 2: Assign \"Alloy Steel\" (density=7730.0) as the material and evaluate the part's Inertia Tensor at the Centroid and at the Origin. Format the result strictly as a JSON object: {\"Inertial_Tensor\": {\"At_Centroid\": <value>, \"At_Origin\": <value>}}, where <value> is the exact numerical array. Save this JSON file to \"C:\\Users\\Docker\\Downloads\\00000915.json.\"

3D Model + Edit

Complete the following tasks sequentially:\nTask 1: Model this part in Inventor: Start by creating a new coordinate system with Euler angles set to zero, zero, and negative ninety degrees, and a translation vector of zero, zero point zero one one two, and zero. Next, draw a two-dimensional sketch on the first face. In the first loop, draw a series of lines starting from the origin (zero, zero) to (zero point zero seven five, zero), then to (zero point zero seven five, zero point zero three seven five), continuing to (zero point six seven five, zero point zero three seven five), then to (zero point six seven five, zero), and finally back to (zero point seven five, zero). From there, draw a line to (zero point seven five, zero point five six two five), then to (zero, zero point five six two five), and back to the origin. For the second loop, start at (zero point zero seven five, zero point one one two five) and draw a line to (zero point six seven five, zero point one one two five), then to (zero point six seven five, zero point four eight seven five), and back to (zero point zero seven five, zero point four eight seven five). Rotate the sketch using the same Euler angles and translate it using the same vector. Extrude the sketch zero point zero one one two units in the direction of the normal, ensuring no extrusion in the opposite direction. Finally, create a new solid body with this extrusion. The dimensions of the base are length zero point seven five, width zero point five six two five, and height zero point zero one one two five. **After creating the solid body, export the model as an STL file to C:/Users/Docker/Downloads/00005144.stl and as a STEP file to C:/Users/Docker/Downloads/00005144.step.**\nTask 2: Based on the Python script you generated in Task 1, modify the script to: Misposition the correctly located corner of the inner pocket so that the lower-right corner is at x = 0.758 and y = 0.1961. Ensure the right-hand edge of the inner pocket is not a vertical line at x = 0.758 that runs between y = 0.1961 and y = 0.4875, producing a distorted rectangular cut with non-horizontal top and bottom edges.\nExport the modified model as an STL file to C:/Users/Docker/Downloads/00005144_level_1_0.stl."

Figure 7: Detailed examples of input instructions across all specific task categories in ComCAD-Bench. The examples highlight the professional engineering terminology, precise geometric constraints, and complex workflows required to complete each automated CAD operation.

Table 12: System prompts used in our evaluation across Baseline, Few-Shot, and RAG settings. Unnecessary JSON arguments and code details are omitted for brevity.

Baseline Prompt

Role: You are an expert CAD engineer and Python programmer specialized in COM interface automation. Your goal is to write executable Python codes that call the COM APIs (`win32com.client`) to programmatically drive CAD software (AutoCAD, Inventor, Solidworks) and complete the given CAD task.

Rules:
- Think before coding. Generate deterministic, non-interactive codes.
- Always output the complete, fully executable Python script. Do NOT provide partial snippets.
- Use only `win32com.client`, `pythoncom`, and Python standard libraries.

Inputs: 1) Task instruction. 2) Terminal output from the last run. 3) Current screenshot after the last run.

Outputs:
1) Brief reasoning inside `<thinking>...</thinking>`.
2) A high-level decision wrapped as: ```` ```decision CODE (or DONE/FAIL) ``` ````
3) If and only if the decision is CODE, output a single ```` ```python ... ``` ```` block.

Few-Shot Prompt (Appended to Baseline)

Example: 3D Modeling in Solidworks
Task Instruction: Model this part in Solidworks: To construct the first part of the cylinder... *[Detailed dimensions and constraints omitted for brevity]* ...export the model as an STL and STEP file.

Output:

````
<thinking>
Create a new part, select the Right Plane... [Reasoning Omitted]
</thinking>
```decision
CODE
```
```python
import win32com.client
# ... [Full Code Omitted] ...
```
````

RAG Prompt (Appended to Baseline)

External Knowledge Context:
Here are some COM APIs that might be useful for completing this task.

```
[
{
"type": "method",
"owner_interface": "ISldWorks",
"name": "NewDocument",
# ... [API details and other APIs Omitted] ...
}
]
```