IterCAD: Iterative Program Repair for CAD Code Generation from Orthographic Views

Yuchuan Wu
ycwu24@m.fudan.edu.cn
Fudan University
Shanghai, China

Ke Niu
kniu22@m.fudan.edu.cn
Fudan University
Shanghai, China

Haiyang Yu
hyyu20@fudan.edu.cn
ByteDance Inc.
Shanghai, China

Zhuofan Chen
zfchen23@m.fudan.edu.cn
Fudan University
Shanghai, China

Xiangyang Xue
xyxue@fudan.edu.cn
Fudan University
Shanghai, China

Bin Li✉
libin@fudan.edu.cn
Fudan University
Shanghai, China

Abstract

Generating executable CAD code from dimension-annotated orthographic drawings is a challenging task requiring geometric understanding, procedural reasoning, and precise numerical prediction. Existing vision-language approaches typically formulate this problem as one-shot generation, preventing the model from inspecting intermediate CAD results and correcting early mistakes, often leading to non-executable code or geometrically inconsistent outputs. In this paper, we propose **IterCAD**, an iterative framework that reformulates orthographic-view-to-CAD generation as a progressive program repair process. Instead of predicting the final CAD code in a single pass, IterCAD repeatedly analyzes the current CAD result, reasons about its discrepancy with the target views, and explicitly decides whether to `REVISE` the code or `STOP` the refinement process. To make iterative repair learnable, we further construct **IterCAD-RS**, a structured revise-or-stop supervision set containing both repairable intermediate CAD states and already-correct states, and develop a three-stage training strategy for initial generation, revision learning, and multi-turn RL optimization. By closing the loop between visual understanding, geometric verification, and code refinement, IterCAD progressively corrects structural and parametric errors. Experiments on CADExpert show that IterCAD consistently improves code executability and geometric fidelity over strong one-shot baselines.

Keywords

Computer-Aided Design, CAD Code Generation, Large Vision-Language Models, Reinforcement Learning

1 Introduction

As a fundamental tool in industrial design and manufacturing, Computer-Aided Design (CAD) [29] has long been essential for the digital creation of engineering products through formalized modeling languages. In modern production pipelines, design concepts are typically translated into precise and editable CAD models before fabrication, simulation, and downstream verification. As a result, automatically generating high-quality CAD code has become an important research problem with both academic significance and practical value.

Compared with directly reconstructing low-level geometric representations such as point clouds, voxels, or meshes, generating parametric CAD code [26, 35] is more aligned with real-world engineering requirements. CAD code is compact, editable, and semantically meaningful: it explicitly encodes the construction logic of a shape, supports downstream modification and validation, and can be directly integrated into industrial design workflows. Therefore, CAD code generation is not merely a 3D reconstruction problem, but a structured code generation task that requires both geometric understanding and procedural reasoning.

From the perspective of practical design workflows, generating CAD code from orthographic engineering drawings is particularly meaningful [33]. In real industrial scenarios, engineers commonly communicate geometric intent through orthographic projections with precise dimension annotations, rather than through free-form natural language descriptions. Such drawings naturally provide the structural and numerical information required for CAD modeling, making them a more realistic and scalable input modality for automatic CAD generation. However, converting dimension-annotated orthographic views into executable CAD code remains highly challenging, since the model must simultaneously infer the underlying 3D geometry, recover the correct sequence of modeling operations, and predict precise numerical parameters.

Existing vision-language approaches [18, 19, 33] have shown promising progress by directly mapping input drawings to CAD code in a one-shot manner. However, this paradigm suffers from an inherent limitation for orthographic-view-to-CAD generation. Generating correct CAD code is a strongly coupled sequential process: an early mistake in workplane selection, operation ordering, geometric topology, or dimensional prediction can propagate through subsequent steps and lead to invalid or geometrically inconsistent results. More importantly, once such an error is made, one-shot methods have no opportunity to inspect the generated result, identify the discrepancy with the target drawing, and revise the code accordingly. As shown in Figure 1, existing one-shot methods may produce either non-executable code or incorrect geometry, while many failures in fact remain repairable if the model is allowed to check the current CAD result and refine it iteratively.

We argue that this limitation is especially severe in the setting of dimension-annotated orthographic-view-to-CAD generation. Unlike tasks where approximate outputs may still be acceptable, CAD code generation demands strict executability and geometric faithfulness. A small deviation in a hole diameter, extrusion depth, or Boolean operation may render the code unusable or significantly

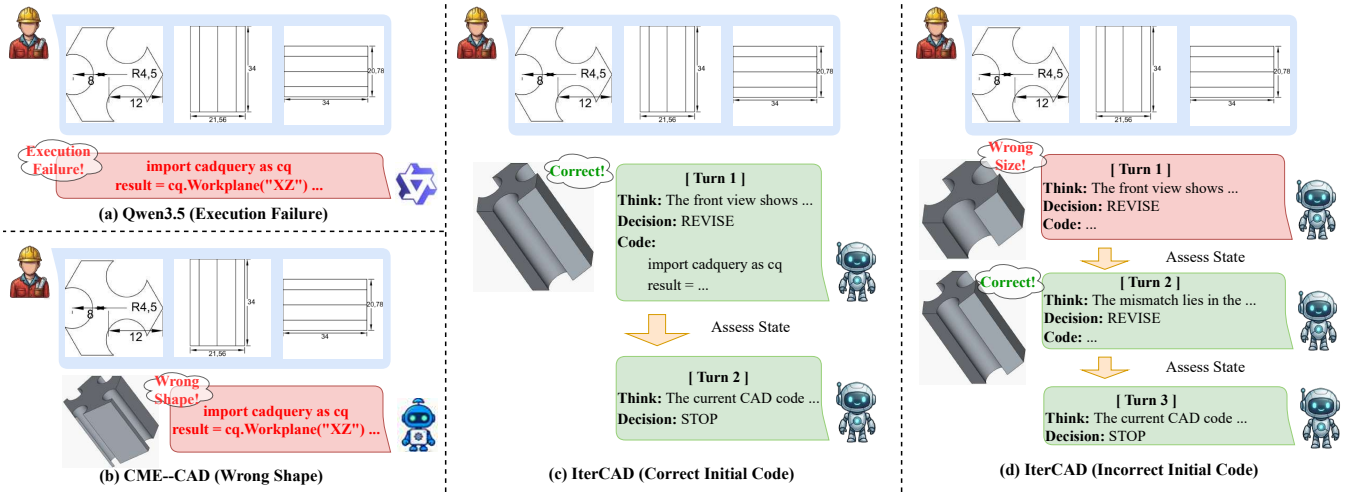

Figure 1: Comparison of one-shot CAD generation and IterCAD. While one-shot methods often fail to produce correct CAD programs, IterCAD progressively improves the program through iterative state assessment and revision.

alter the final shape. At the same time, CAD code generation also presents a favorable property for iterative refinement: intermediate outputs take the form of executable CAD code that can be inspected, compared against the target views, and further revised. This suggests that CAD code generation should not be treated as a single-pass prediction problem, but rather as a progressive refinement process in which the model can repeatedly examine its current result and correct its own mistakes.

Based on this insight, we propose **IterCAD**, an iterative framework for CAD code generation from dimension-annotated orthographic drawings. Instead of producing the final answer in one step, **IterCAD** enables the model to repeatedly analyze the current CAD result, reason about its mismatch with the target views, and explicitly decide whether to `REVISE` the code or `STOP` the refinement process. To make such iterative repair learnable, we further construct **IterCAD-RS**, a structured revise-or-stop supervision set, and develop a dedicated three-stage training recipe for initial generation, revision learning, and multi-turn policy optimization. By closing the loop between visual understanding, geometric verification, and code revision, **IterCAD** progressively corrects both structural and parametric errors, leading to more accurate and executable CAD reconstruction.

Different from simply eliciting longer reasoning traces, **IterCAD** introduces a structured iterative paradigm tailored to the characteristics of CAD modeling. It allows the model to recover from erroneous initial generations, refine nearly correct CAD code with subtle dimensional inconsistencies, and terminate the process explicitly once sufficient consistency has been achieved. Extensive experiments demonstrate that **IterCAD** consistently improves both executability and geometric accuracy over strong one-shot baselines.

Our main contributions are summarized as follows:

- We reformulate dimension-annotated orthographic-view-to-CAD generation as an iterative program repair problem, in which the model operates on executable intermediate CAD states and learns to make explicit `REVISE`/`STOP` decisions. To support this formulation, we construct **IterCAD-RS**, a structured revise-or-stop supervision set containing both repairable CAD states paired with `REVISE` targets and already-correct CAD states paired with `STOP` targets.
- We propose **IterCAD**, a unified iterative framework that integrates discrepancy-aware reasoning, explicit revise-or-stop decision-making, and iterative CAD code refinement in a closed loop. Building on this framework, we develop a dedicated three-stage training recipe that progressively teaches the model initial CAD generation, intermediate-state revision, and multi-turn self-repair policy optimization.
- Extensive experiments on CADExpert [18] show that **IterCAD** consistently outperforms strong one-shot baselines in both code executability and geometric fidelity. Further analyses verify the effectiveness of the iterative program repair formulation, **IterCAD-RS**, and the proposed three-stage training strategy.

2 Related Work

2.1 CAD Code Generation from Visual Inputs

Large vision-language models have demonstrated substantial value across diverse application domains [9, 10, 20, 21, 37–39]. Building on these advances, recent research has increasingly explored their potential for CAD code generation. CAD code generation aims to produce executable and editable 3D design programs directly from user inputs, such as text descriptions, images, or engineering drawings [2, 3, 11, 22, 36]. Compared with general code generation, this task requires not only structured program prediction but also accurate spatial reasoning, geometric consistency, and numerical precision. Even small structural or parametric errors may lead to non-executable programs or geometries that deviate substantially from the target design.

Recent studies have explored CAD generation from visual inputs under different formulations. CAD-MLLM [36] and GenCAD [2]

Table 1: Perturbation taxonomy used to construct IterCAD-RS. Starting from a correct structured CAD state (*ShapeSpec*), we inject one or two geometric perturbations to synthesize executable but geometrically inconsistent intermediate states for revise-or-stop supervision.

Structural Family	Perturbation Target	Representative Operations	Typical Resulting Errors
Body-level	Sketch plane	Change `XY`/`YZ`/`XZ` workplane	Wrong global orientation or projection mismatch
Body-level	Base profile geometry	Modify primitive type, polygon side count, or profile scale	Incorrect outer silhouette or overall geometry
Body-level	Extrusion parameter	Modify extrusion height or depth	Incorrect thickness or global dimensions
`cut`	Inner profile geometry	Change inner profile type or size	Wrong cavity shape or incorrect inner dimensions
`cut`	Modifier existence	Remove the cut operation	Missing inner structure
`hole_pattern`	Hole multiplicity	Increase or decrease the number of holes	Incorrect hole count
`hole_pattern`	Hole layout	Modify radial offset or distribution radius	Wrong hole arrangement
`hole_pattern`	Hole size / existence	Change hole diameter or remove the pattern	Incorrect or missing hole structure
`edge_finish`	Finish type	Switch between `fillet` and `chamfer`	Incorrect edge treatment
`edge_finish`	Finish magnitude / existence	Modify finish value or remove the operation	Wrong or missing local edge detail
Topology-level	Modifier insertion	Add an extra `cut`, `hole_pattern`, or `edge_finish`	Spurious geometric structure

use vision-language models to directly translate visual inputs into CAD commands or programs, enabling end-to-end visual-to-code generation. Img2CAD [6] adopts a two-stage framework that decouples structure prediction from parameter regression, improving flexibility in program synthesis. CAD2Program [33] introduces a more expressive code representation to better support complex CAD construction. CAD-Llama [14] and CAD-Coder [11] further improve structured parametric CAD generation through hierarchical annotations or expert-designed intermediate descriptions.

Beyond direct visual-to-code prediction, some works have begun to incorporate verification or improvement mechanisms into CAD generation. In particular, CADCodeVerify [3] introduces an automated verification-and-improvement pipeline in which a pretrained vision-language model inspects the rendered CAD result, generates and answers validation questions, and feeds the resulting feedback back to the generator for refinement. This line of work shows that CAD generation can benefit from going beyond pure one-shot prediction, especially since generated CAD programs are structured, executable, and amenable to downstream checking.

Nevertheless, existing approaches still mainly treat refinement as an external feedback step applied to the generated final program, rather than explicitly modeling CAD generation itself as a learned multi-turn repair process over intermediate executable states. In contrast to CADCodeVerify, which relies on prompting-based automated verification at inference time, our method adopts a unified iterative formulation in which the model directly operates on intermediate CAD states, progressively revises them, and explicitly learns a revise-or-stop policy for adaptive refinement.

2.2 Reinforcement Learning in CAD Generation

Reinforcement learning [13, 16, 24, 25, 30, 31, 34] has recently emerged as a promising direction for improving CAD code generation beyond standard supervised fine-tuning [17]. Instead of relying solely on next-token prediction, RL-based methods optimize generated CAD programs with task-specific objectives, such as executability, geometric accuracy, and alignment with design intent.

Among existing works, CAD-RL [19] is a representative effort in this direction. It introduces a multimodal chain-of-thought-guided reinforcement learning framework for precise CAD code generation, showing that reward-driven optimization can substantially improve the numerical accuracy, executability, and reasoning quality of generated CadQuery programs. This line of work highlights the value of incorporating execution-aware and geometry-aware feedback into CAD generation.

However, existing RL-based CAD generation methods still primarily optimize the quality of the final generated program, rather than explicitly modeling generation as a multi-turn repair process over intermediate executable CAD states. In contrast, our method focuses on iterative program revision, where the model repeatedly inspects the current CAD result, decides whether to `REVISE` or `STOP`, and performs adaptive multi-step correction accordingly.

3 Iterative Repair Formulation and IterCAD-RS Construction

Orthographic-view-to-CAD generation is naturally suited to an iterative program repair paradigm, since intermediate CAD code is executable and can be further revised based on discrepancies with the target orthographic views. In this section, we first formulate the task as a multi-turn revise-or-stop process, and then describe the construction of **IterCAD-RS**, a structured revise-or-stop supervision set for iterative program repair.

3.1 Multi-turn Program Repair Formulation

We formulate orthographic-view-to-CAD generation as an iterative program repair process rather than a one-shot prediction problem. Given the input orthographic views x, the model progressively refines CAD code over multiple turns until it decides to stop. The initial state is defined as

$$s_0 = (x, \texttt{EMPTY}), \tag{1}$$

where no CAD code has been generated yet. At turn t, the state is defined as

$$s_t = (x, h_{t-1}), \tag{2}$$

where h_{t-1} denotes the interaction history available before turn t. In our setting, this history contains the model outputs from previous turns, including the generated `<think>` rationale, the `<decision>`, and the current cad `<code>`. Therefore, the model does not merely

revise the latest CAD code in isolation, but performs the next repair step by conditioning on its own earlier reasoning process, decisions, and current code state.

At each turn t, the model outputs a structured response

$$o_t = (r_t, a_t, y_t), \tag{3}$$

where r_t denotes the <think> rationale, $a_t \in \{\texttt{REVISE}, \texttt{STOP}\}$ is the decision in the <decision> field, and y_t is the CAD code in the <code> field. The interaction history is updated as

$$h_t = h_{t-1} \oplus o_t, \tag{4}$$

where $\oplus$ denotes appending the current-turn output to the history. If $a_t = \texttt{REVISE}$, the updated CAD code y_t is used as the input state for the next turn; if $a_t = \texttt{STOP}$, the iterative process terminates and y_t is taken as the final prediction. The trajectory terminates when

$$a_t = \texttt{STOP} \quad \text{or} \quad t = T_{\max}. \tag{5}$$

This formulation highlights several important properties of our setting. First, intermediate outputs are structured and executable intermediate states rather than transient text tokens, and can therefore support subsequent refinement. Second, stopping is modeled as a learned action instead of a hand-crafted post-processing rule, enabling the model to adapt the number of refinement steps to the difficulty of each sample. More broadly, this formulation internalizes the core steps of CAD generation—reasoning over the input views, evaluating the current code state, deciding whether further correction is needed, and producing the next revision—within a single unified model, rather than relying on external verification tools or manually designed control heuristics. In this sense, IterCAD goes beyond one-shot CAD code prediction toward a more adaptive and self-refining paradigm for CAD code generation.

3.2 IterCAD-RS: Structured Revise-or-Stop Supervision for Iterative Program Repair

A key supervision bottleneck for iterative program repair is that standard orthographic-view-to-CAD datasets provide only the final correct CAD code, but not the intermediate states needed for learning revise-or-stop behavior. To address this, we construct **IterCAD-RS**, a structured supervision set built on top of the SFT training split of CADExpert [18], which augments each ground-truth sample with both repairable intermediate states and positive stopping states.

Specifically, instead of perturbing raw code strings directly, we first convert each correct CadQuery program into a structured geometric state representation, denoted as *ShapeSpec*, and then apply controlled semantic perturbations in this space to synthesize executable but geometrically inconsistent intermediate states. These perturbations cover both body-level and modifier-level attributes, producing realistic errors that remain executable while deviating from the target orthographic views, as summarized in Table 1. We further apply validity-preserving filtering to remove degenerate or unchanged cases.

For each synthesized wrong state, we pair it with the original correct target and automatically generate a short repair description, denoted as *fix_text*, which serves as supervision for the <think> field. This yields revision samples of the form

$$(\text{views}, \text{wrong code}) \rightarrow (\text{repair rationale}, \texttt{REVISE}, \text{correct code}),$$

while correct or semantically equivalent executable states are used to construct STOP supervision. In this way, IterCAD-RS provides explicit supervision for both intermediate revision and termination behavior. More details on perturbation design, filtering rules, and data statistics are provided in the supplementary material.

4 Methodology

4.1 Overview of IterCAD

As illustrated in Figure 2, **IterCAD** formulates orthographic-view-to-CAD generation as an iterative program repair process rather than a one-shot prediction problem. The overall framework consists of three training stages. In Stage I, the model learns to generate an initial CAD draft from dimension-annotated orthographic views, establishing the basic vision-to-code mapping. In Stage II, the model learns revise-or-stop behavior from intermediate CAD states: given the target views together with a current CAD state, it reasons about the mismatch between the current CAD code and the target specification, and outputs either a STOP decision if the current code is already satisfactory or a REVISE decision followed by updated CAD code otherwise. In Stage III, we further optimize the full multi-turn repair process with GRPO over sampled repair trajectories. While the first two stages equip the model with the basic capabilities for initial generation and intermediate revision, they do not by themselves ensure that the model can effectively coordinate these abilities over a multi-turn process. By optimizing over diverse sampled trajectories, Stage III encourages the model to jointly leverage prior generation and revision knowledge, learn when additional refinement is truly beneficial, and decide when the current state is sufficient to terminate.

4.2 Training Data

The three training stages of **IterCAD** use different but complementary forms of supervision, all built upon CADExpert [18], an industrial-oriented benchmark for executable and editable CAD code generation. Each sample in CADExpert contains dimension-annotated orthographic views together with the corresponding executable CADQuery code, providing the basic supervision for orthographic-view-to-CAD learning.

In Stage I, we directly use 8,960 standard orthographic-view-to-CAD pairs from the SFT training split of CADExpert to train the model for initial CAD code generation. In Stage II, we further construct **IterCAD-RS** on top of this split, yielding 22,000 structured revise-or-stop supervision samples derived from intermediate CAD states. In Stage III, we use an additional 4,480 training samples for multi-turn RL optimization. The data used in Stage III is disjoint from that used in Stages I and II.

Overall, the training data progresses from direct generation supervision, to structured revise-or-stop supervision, and finally to trajectory-level optimization. More details on data construction are provided in the supplementary material.

4.3 Three-stage Training Recipe

As illustrated in Figure 2, **IterCAD** is trained with a three-stage recipe rather than end-to-end joint optimization from scratch. This

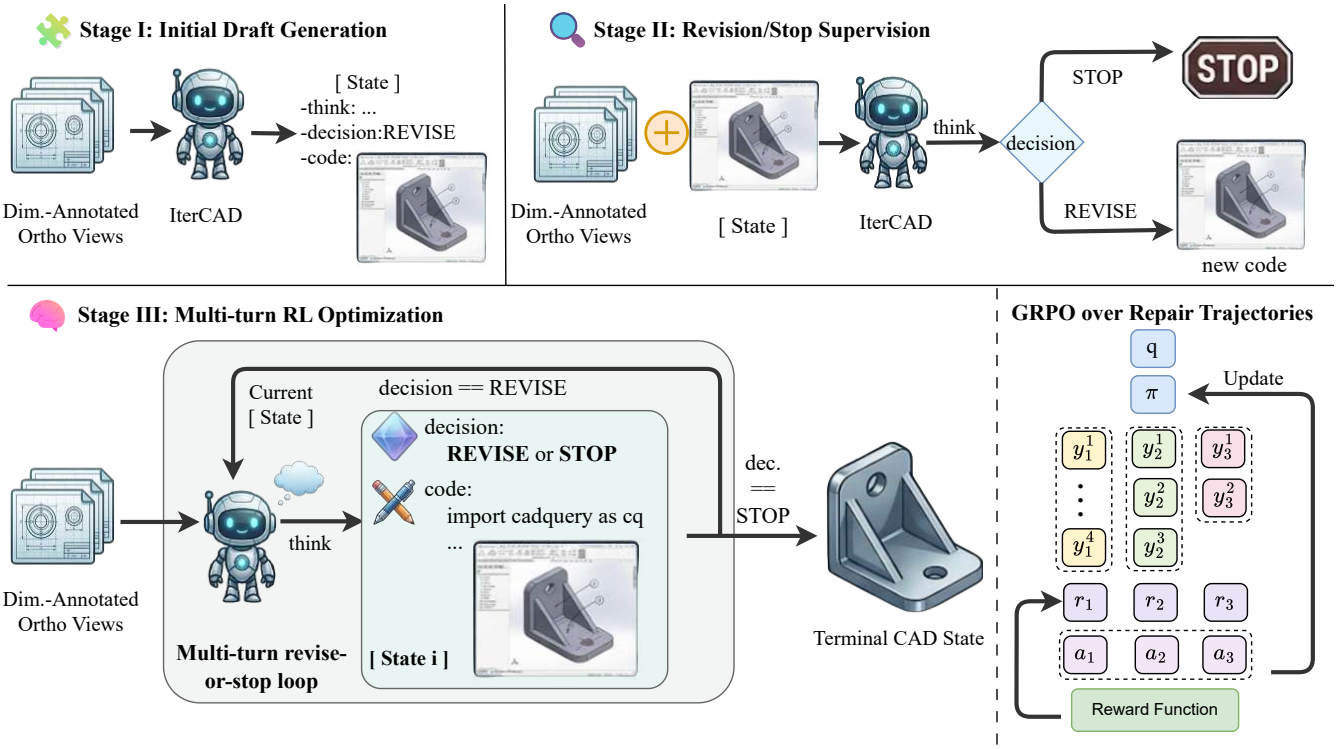

Figure 2: Overview of IterCAD. Stage I learns first-turn initial draft generation from dimension-annotated orthographic views. Stage II learns structured revise-or-stop behavior from intermediate CAD states. Stage III further optimizes the full multi-turn repair process with GRPO over sampled repair trajectories, yielding a unified iterative policy for initial generation, revision, and adaptive stopping.

design is motivated by the observation that initial CAD draft generation, revise-or-stop decision making, and full multi-turn self-correction are related but distinct capabilities. While the first two stages provide the model with the basic abilities for initial generation and intermediate repair, they do not by themselves ensure effective coordination of these abilities during iterative inference. Therefore, we progressively train the model from supervised initial drafting, to structured revise-or-stop learning, and finally to trajectory-level RL optimization, yielding a more stable and effective training process for iterative program repair.

4.3.1 Stage I: Initial Draft Generation. In Stage I, we train the model to produce the first-turn structured response from the input orthographic views. Different from standard one-shot CAD code generation, the model does not directly output only the final CAD code. Instead, it is trained under the same interaction format used in later stages, where the current CAD state is initialized as empty and the model performs the first revision step accordingly.

Formally, given the input orthographic views x and an empty initial code state, the model predicts

$$(x, \texttt{EMPTY}) \rightarrow (r_1, \texttt{REVISE}, y_1), \tag{6}$$

where r_1 denotes the <think> rationale and y_1 denotes the CAD code in the <code> field. Since no valid CAD code exists at the first turn, the decision is always set to REVISE, and the model is required to generate an initial executable CAD draft.

This stage serves two purposes. First, it establishes the basic vision-to-code capability from dimension-annotated orthographic views. Second, by aligning the output format with the later revise-or-stop stages, it provides a consistent initialization for iterative program repair, so that subsequent stages can directly build on the same structured interaction pattern.

4.3.2 Stage II: Revision/Stop Supervision. In Stage II, we train the model with structured revise-or-stop supervision on intermediate CAD states. Given the target orthographic views x together with the interaction history h_{t-1}, the model predicts a structured output

$$(x, h_{t-1}) \rightarrow (r_t, a_t, y_t), \tag{7}$$

where r_t denotes the <think> rationale, $a_t \in \{\texttt{REVISE}, \texttt{STOP}\}$ is the decision in the <decision> field, and y_t denotes the CAD code in the <code> field. Here, h_{t-1} contains the model outputs from previous turns, including the generated rationale, decision, and current CAD code. Therefore, the model does not revise the current code in isolation, but predicts the next action by conditioning on its own earlier reasoning process, decisions, and code state.

The supervision used in this stage is provided by **IterCAD-RS**, which contains two complementary types of samples: revision samples, where executable but geometrically inconsistent intermediate

states are paired with repair rationales, REVISE decisions, and corrected CAD code; and stopping samples, where already-correct or semantically equivalent executable states are paired with STOP decisions. In this way, Stage II teaches the model not only how to perform targeted correction when geometric discrepancies exist, but also how to decide when the current state is already sufficient to terminate.

Importantly, the training targets preserve the same structured output protocol used at inference time, namely <think>, <decision>, and <code>. As a result, Stage II equips the model with the core abilities required for iterative program repair: interpreting intermediate states, producing explicit revise-or-stop decisions, and generating updated CAD code in a unified format.

4.3.3 Stage III: Multi-turn RL Optimization. After the first two supervised stages, the model has acquired the basic abilities required for iterative CAD repair, including initial draft generation, intermediate-state revision, and explicit stopping. However, supervised training alone does not guarantee that the model can effectively coordinate these abilities during multi-turn inference. In particular, while Stage I and Stage II teach the model how to generate, revise, and stop under supervised targets, they do not by themselves ensure that these behaviors will be integrated into a coherent repair policy over full trajectories.

To address this, in Stage III we further optimize the complete multi-turn repair process using GRPO [27]. Starting from an initial CAD draft, the model interacts with its own intermediate CAD states and produces a repair trajectory

$$\tau = (s_0, o_1, s_1, \ldots, o_T, s_T), \tag{8}$$

where each turn output o_t follows Eq. (3). The Stage III objective is to maximize the expected trajectory-level reward

$$\max_{\pi_\theta} \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right], \tag{9}$$

where $R(\tau)$ evaluates the overall quality of the multi-turn repair process.

A key property of this stage is that optimization is performed over sampled multi-turn trajectories instead of isolated single-step corrections. This is important in our setting, since **IterCAD** is intended to learn a revise-or-stop policy over executable intermediate CAD states, rather than a final-step correction heuristic. By optimizing Eq. (9) over diverse sampled trajectories, GRPO encourages the model to jointly leverage the generation and revision knowledge acquired in the previous stages, learn when further refinement is beneficial, and decide when the current state is already sufficient to terminate. As a result, Stage III turns the model from a supervised reviser into a true multi-turn iterative repair policy.

4.4 Reward Design

To optimize **IterCAD** under the multi-turn revise-or-stop setting, we design a composite reward that combines final CAD quality, structural validity, and decision correctness. Since the last turn may output STOP without providing a new <code> block, we evaluate the trajectory using an *effective final code*, i.e., the CAD code corresponding to the terminal program state. If the last turn outputs REVISE with a valid <code> block, we use that code directly; otherwise, we trace back to the most recent valid CAD code in the trajectory history.

Based on the effective final code, the overall reward is defined as

$$R = R_{\text{final}} + \lambda_f R_{\text{format}} + \lambda_s R_{\text{syntax}} + \lambda_d R_{\text{decision}}, \tag{10}$$

where $\lambda_f = 0.10$, $\lambda_s = 0.05$, and $\lambda_d = 0.20$. Here, R_{final} serves as the dominant learning signal, while the remaining terms act as auxiliary regularizers.

Specifically, R_{format} measures whether the model follows the required structured interaction protocol, encouraging stable outputs in the <think>, <decision>, and <code> format. R_{syntax} focuses on code validity, rewarding syntactically well-formed executable code while serving only as a lightweight regularizer. R_{decision} focuses on revise-or-stop behavior, encouraging the model to stop only when the current CAD state is already sufficiently good and discouraging premature stopping or invalid decisions. Finally, R_{final} evaluates the geometric quality of the effective final code and remains the primary optimization target throughout training.

Overall, this reward is outcome-centric but behavior-aware: it prioritizes the correctness of the terminal CAD state while also regularizing the model to produce valid structured outputs and make appropriate stopping decisions. More detailed definitions and implementation details are provided in the supplementary material.

Table 2: Main results on CADExpert. IterCAD-Q3VL and IterCAD-Q3.5 denote our method instantiated with Qwen3-VL-8B-Instruct and Qwen3.5-9B, respectively. Best results are shown in bold, and second-best results are underlined.

Model	IoU (%) ↑	Mean CD ↓	Med CD ↓	Exec. (%) ↑
LLaVA-1.5 [15]	0.73	29.36	8.14	4.79
Phi-3.5-Vision [1]	3.44	27.92	7.75	8.50
InternVL2.5 [7]	11.98	22.27	7.36	23.92
Qwen2.5-VL [5]	19.21	20.68	6.64	31.27
InternVL3 [40]	24.59	21.40	6.62	29.68
Gemini2.5 Pro [8]	30.86	14.13	5.97	39.40
GPT5-Mini [28]	35.15	9.89	4.81	47.13
Doubao-1.6 [12]	35.62	7.54	4.35	52.89
Qwen3-VL [4]	37.04	6.96	3.84	54.79
Qwen3.5 [23]	43.91	5.36	3.17	59.61
CAD-RL [19]	71.84	1.38	0.36	97.32
CME-CAD [18]	80.71	1.00	0.11	<u>98.25</u>
IterCAD-Q3VL	<u>85.10</u>	<u>0.8564</u>	<u>0.1069</u>	97.31
IterCAD-Q3.5	**91.61**	**0.5387**	**0.1038**	**99.33**

5 Experiments

5.1 Dataset and Metrics

We conduct all experiments on CADExpert [18], an industrial-oriented benchmark for executable and editable CAD code generation, following its standard benchmark setting for orthographic-view-to-CAD generation.

Following prior work [18], we evaluate model performance using four complementary metrics: Intersection-over-Union (IoU), Mean Chamfer Distance (Mean CD), Median Chamfer Distance (Med CD), and Executability (Exec.). IoU measures volumetric overlap between

predicted and reference shapes, Mean CD and Med CD measure geometric discrepancy in point cloud space, and Executability measures whether the generated CAD code can be successfully executed. Higher values are better for IoU and Exec., while lower values are better for Mean CD and Med CD.

5.2 Implementation Details

We evaluate **IterCAD** on two backbone models: **Qwen3.5-9B** [23] and **Qwen3-VL-8B-Instruct** [32]. For both backbones, we adopt the same three-stage training recipe, where the Stage I checkpoint initializes Stage II, and the Stage II checkpoint further initializes Stage III. Stage I and Stage II are trained with supervised fine-tuning, while Stage III is optimized with GRPO. Unless otherwise specified, all three stages use a learning rate of 1×10^{-5} and are trained for one epoch.

For Stage III, we use 4 rollouts per training sample, and set the maximum number of refinement turns to 4. The same rollout setting is used for both backbone models.

At inference time, the model follows the same iterative repair protocol as in training. Given the input orthographic views, it first produces an initial CAD draft and then iteratively predicts either `REVISE` or `STOP`. The process terminates when the model outputs `STOP` or when the maximum number of refinement rounds reaches 4. No external stopping heuristic is used. For the main benchmark results, both backbone models are evaluated under the same training and inference pipeline.

5.3 Main Results

Table 2 reports the main results on CADExpert. **IterCAD** achieves the best overall performance, with IterCAD-Q3.5 ranking first on all four metrics. Specifically, it reaches 91.61% IoU, 0.5387 Mean CD, 0.1038 Med CD, and 99.33% executability, establishing a new state of the art for orthographic-view-to-CAD generation.

Compared with the strong multi-expert baseline CME-CAD, IterCAD-Q3.5 improves IoU from 80.71% to 91.61%, reduces Mean CD from 1.00 to 0.5387, and increases executability from 98.25% to 99.33%. While CME-CAD strengthens one-shot generation through expert decomposition, IterCAD uses a unified model to iteratively inspect and repair executable intermediate CAD states. The substantial improvement therefore suggests that explicit self-correction is more effective than strengthening single-pass prediction alone.

This conclusion is further supported by IterCAD-Q3VL. Despite using the lighter Qwen3-VL-8B-Instruct backbone, it surpasses CME-CAD on all geometric metrics, achieving 85.10% IoU, 0.8564 Mean CD, and 0.1069 Med CD while maintaining high executability. The consistent gains across two backbones indicate that the improvement primarily comes from the iterative repair formulation rather than a particular model architecture. This robustness is particularly important because the two backbones differ in capacity and pretraining, yet benefit from the same repair strategy.

Compared with general-purpose pretrained VLMs, both specialized CAD methods retain a large advantage, confirming the need for task-specific modeling of geometry, code structure, and executability. Moreover, the gains span both geometric overlap and execution reliability, indicating that iterative refinement improves not only accuracy but also practical usability. Overall, IterCAD improves geometric fidelity without sacrificing code validity, supporting multi-turn repair as an effective alternative to one-shot CAD code generation.

Table 3: Ablation study of IterCAD on CADExpert. S1, S2, and S3 denote Stages I, II, and III, respectively, and MT denotes multi-turn inference. Best results are shown in bold.

S1	S2	S3	MT	IoU (%) ↑	Mean CD ↓	Med CD ↓	Exec. (%) ↑
Base model: Qwen3.5-9B							
✓	×	×	×	57.42	3.5819	0.1939	96.39
✓	✓	×	×	54.46	3.4506	0.3520	93.65
✓	✓	×	✓	60.40	2.8102	0.1590	93.37
✓	×	✓	✓	64.26	2.6319	0.1448	94.28
✓	✓	✓	×	89.91	0.7492	0.1193	98.51
✓	✓	✓	✓	**91.61**	**0.5387**	**0.1038**	**99.33**
Base model: Qwen3-VL-8B-Instruct							
✓	×	×	×	37.37	5.1648	3.9749	85.48
✓	✓	×	✓	28.30	5.8017	5.4882	79.87
✓	✓	✓	×	83.17	1.1057	0.1493	96.38
✓	✓	✓	✓	**85.10**	**0.8564**	**0.1069**	**97.31**

5.4 Ablation Studies

Table 3 reports the ablation results of **IterCAD** on CADExpert. The full configuration consistently achieves the best performance on both backbones, showing that the three-stage training recipe and multi-turn inference are all important to the final gains.

A notable observation is that the improvement cannot be explained simply by using more supervision. For example, adding Stage II does not directly improve performance over Stage I under single-turn inference, and can even hurt both accuracy and executability. This suggests that Stage II introduces a more structured but also more challenging learning problem: it teaches the model to interpret intermediate CAD states and predict revise-or-stop behavior, but these abilities are not yet fully utilized when inference is restricted to a single pass.

Importantly, once multi-turn inference is enabled, the Stage I+II model becomes clearly stronger than its single-turn counterpart. On Qwen3.5-9B, adding multi-turn inference to the Stage I+II setting improves IoU from 54.46 to 60.40 while also reducing both mean and median Chamfer Distance. This result provides early evidence for the validity of our formulation: although Stage II alone does not immediately improve one-shot generation, it equips the model with intermediate-state revision ability that becomes useful when the model is actually allowed to refine its prediction across turns.

Stage III is therefore crucial. It turns locally supervised generation and revision abilities into a coherent trajectory-level repair policy. This effect is already visible in the Stage I+III setting on Qwen3.5-9B, which outperforms Stage I alone, indicating that trajectory-level optimization can already activate useful repair behavior. However, the full model still performs much better, showing that Stage II and Stage III are complementary: Stage II provides explicit supervision on intermediate repair, while Stage III teaches the model how to coordinate generation, revision, and stopping across turns.

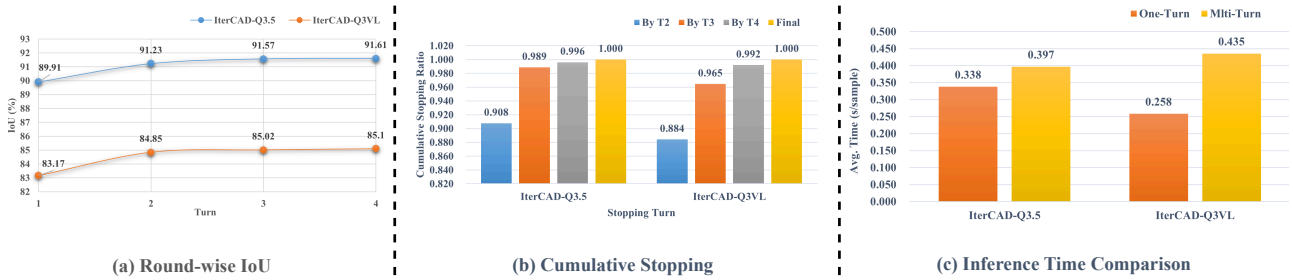

Figure 3: Iterative repair behavior on CADExpert. (a) Round-wise IoU. (b) Cumulative stopping ratios by refinement turn. (c) Average inference time under one-turn and multi-turn settings.

Finally, comparing the full model with and without multi-turn inference shows that the learned iterative policy remains useful at test time. Enabling multi-turn inference consistently improves both geometric metrics and executability on the two backbones. This confirms that **IterCAD** does not benefit only from better training, but also from refining intermediate CAD states during inference.

5.5 Multi-turn Inference Analysis

To better understand how **IterCAD** behaves at test time, we analyze the quality progression, stopping behavior, and inference efficiency of multi-turn inference in Figure 3. Several clear patterns can be observed.

First, iterative refinement consistently improves geometric quality across turns. As shown in Figure 3(a), IoU increases monotonically for both backbones as the refinement proceeds. For IterCAD-Q3.5, IoU improves from 89.91 at Turn 1 to 91.61 at Turn 4, while for IterCAD-Q3VL it increases from 83.17 to 85.10. Most of the gain is obtained in the early turns, especially from Turn 1 to Turn 2, while later turns provide smaller but still consistent improvements. This indicates that the model is able to correct major errors early and then make finer adjustments in later rounds.

Second, the stopping behavior is highly adaptive rather than uniform. Figure 3(b) shows that most samples terminate within the first two or three turns. For IterCAD-Q3.5, 90.8% of samples stop by Turn 2 and 98.9% by Turn 3; for IterCAD-Q3VL, the corresponding ratios are 88.4% and 96.5%. Only a very small fraction of samples require all four turns. This result suggests that the learned stopping policy is effective: the model does not simply run for the maximum number of steps, but instead adjusts the refinement depth according to the difficulty of the current sample.

Finally, the additional cost of multi-turn inference remains moderate. As shown in Figure 3(c), the average inference time increases from 0.338 s to 0.397 s per sample for IterCAD-Q3.5, and from 0.258 s to 0.435 s for IterCAD-Q3VL, when switching from one-turn to multi-turn inference. Combined with the cumulative stopping statistics, this shows that the computational overhead of iterative repair is bounded in practice, since most samples terminate early. Overall, these results indicate that IterCAD achieves a favorable trade-off between refinement quality and inference efficiency: multi-turn inference yields consistent gains, while adaptive stopping prevents unnecessary computation.

More qualitative examples of the iterative repair process, including step-by-step reasoning and CAD refinement trajectories, are provided in the supplementary material.

6 Conclusion

In this paper, we presented **IterCAD**, an iterative program repair framework for CAD code generation from dimension-annotated orthographic views. Instead of treating orthographic-view-to-CAD generation as a one-shot prediction problem, IterCAD reformulates it as a multi-turn revise-or-stop process over executable intermediate CAD states. To make such iterative repair learnable, we further introduced **IterCAD-RS**, a structured revise-or-stop supervision set, together with a three-stage training recipe for initial draft generation, intermediate revision learning, and multi-turn RL optimization.

Experiments on CADExpert show that IterCAD consistently improves both geometric fidelity and code executability over strong baselines. Further ablation and multi-turn inference analyses verify that these gains come from the coordinated effect of structured intermediate supervision, trajectory-level optimization, and adaptive stopping. Overall, our results suggest that orthographic-view-to-CAD generation is better modeled as an iterative repair problem than as a one-shot generation task, and highlight the value of self-refining generation for executable CAD modeling.

Acknowledgments

This work was supported by the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning (Grant No. GZ2022001).

References

[1] Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes,

Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. arXiv:2404.14219 [cs.CL] https://arxiv.org/abs/2404.14219

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

[3] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. 2024. Generating cad code with vision-language models for 3d designs. *arXiv preprint arXiv:2410.05340* (2024).

[4] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631* (2025).

[5] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. 2025. Qwen2.5-VL Technical Report. arXiv:2502.13923 [cs.CV] https://arxiv.org/abs/2502.13923

[6] Tianrun Chen, Chunan Yu, Yuanqi Hu, Jing Li, Tao Xu, Runlong Cao, Lanyun Zhu, Ying Zang, Yong Zhang, Zejian Li, et al. 2025. Img2cad: Conditioned 3-d cad model generation from single image with structured visual geometry. *IEEE Transactions on Industrial Informatics* (2025).

[7] Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. 2024. Expanding Performance Boundaries of Open-Source Multimodal Models with Model, Data, and Test-Time Scaling. *arXiv preprint arXiv:2412.05271* (2024).

[8] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261* (2025).

[9] Teng Fu, Xiaocong Wang, Haiyang Yu, Ke Niu, Bin Li, and Xiangyang Xue. 2023. Denoising-mot: Towards multiple object tracking with severe occlusions. In *Proceedings of the 31st ACM International Conference on Multimedia*. 2734–2743.

[10] Teng Fu, Mengyang Zhao, Ke Niu, Kaixin Peng, and Bin Li. 2026. OmniPT: Unleashing the Potential of Large Vision Language Models for Pedestrian Tracking and Understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 4031–4039.

[11] Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. 2025. CAD-Coder: Text-to-CAD Generation with Chain-of-Thought and Geometric Reward. *arXiv preprint arXiv:2505.19713* (2025).

[12] Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, et al. 2025. Seed1. 5-vl technical report. *arXiv preprint arXiv:2505.07062* (2025).

[13] Weitao Jia, Jinghui Lu, Haiyang Yu, Siqi Wang, Guozhi Tang, An-Lan Wang, Weijie Yin, Dingkang Yang, Yuxiang Nie, Bin Shan, Hao Feng, Irene Li, Kun Yang, Han Wang, Jingqun Tang, Teng Fu, Changhong Jin, Chao Feng, Xiaohui Lv, and Can Huang. 2025. MEML-GRPO: Heterogeneous Multi-Expert Mutual Learning for RLVR Advancement. arXiv:2508.09670 [cs.AI] https://arxiv.org/abs/2508.09670

[14] 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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18563–18573.

[15] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved Baselines with Visual Instruction Tuning. arXiv:2310.03744 [cs.CV] https://arxiv.org/abs/2310.03744

[16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PmLR, 1928–1937.

[17] Ke Niu, Zhuofan Chen, Haiyang Yu, Yuwen Chen, Teng Fu, Mengyang Zhao, Bin Li, and Xiangyang Xue. 2025. Creft-cad: Boosting orthographic projection reasoning for cad via reinforcement fine-tuning. *arXiv preprint arXiv:2506.00568* (2025).

[18] Ke Niu, Haiyang Yu, Zhuofan Chen, Zhengtao Yao, Weitao Jia, Xiaodong Ge, Jingqun Tang, Benlei Cui, Bin Li, and Xiangyang Xue. 2025. Cme-cad: Heterogeneous collaborative multi-expert reinforcement learning for cad code generation. *arXiv preprint arXiv:2512.23333* (2025).

[19] Ke Niu, Haiyang Yu, Zhuofan Chen, Mengyang Zhao, Teng Fu, Bin Li, and Xiangyang Xue. 2026. From intent to execution: Multimodal chain-of-thought reinforcement learning for precise cad code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 8160–8167.

[20] Ke Niu, Haiyang Yu, Mengyang Zhao, Teng Fu, Siyang Yi, Wei Lu, Bin Li, Xuelin Qian, and Xiangyang Xue. 2025. ChatReID: Open-ended Interactive Person Retrieval via Hierarchical Progressive Tuning for Vision Language Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 24245–24254.

[21] Kaixin Peng, Mengyang Zhao, Haiyang Yu, Teng Fu, and Bin Li. 2025. Interpretable oracle bone script decipherment through radical and pictographic analysis with lvlms. *arXiv preprint arXiv:2508.10113* (2025).

[22] Feiwei Qin, Shichao Lu, Junhao Hou, Changmiao Wang, Meie Fang, and Ligang Liu. 2025. Drawing2CAD: Sequence-to-Sequence Learning for CAD Generation from Vector Drawings. In *Proceedings of the 33rd ACM International Conference on Multimedia*. 10573–10582.

[23] Qwen Team. 2026. Qwen3.5: Towards Native Multimodal Agents. https://qwen.ai/blog?id=qwen3.5

[24] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International conference on machine learning*. PMLR, 1889–1897.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[26] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. 2021. Vitruvion: A generative model of parametric cad sketches. *arXiv preprint arXiv:2109.14124* (2021).

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

[28] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. 2025. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267* (2025).

[29] Yuewan Sun, Xingang Li, and Zhenghui Sha. 2025. Large language models for computer-aided design fine tuned: Dataset and experiments. *Journal of Mechanical Design* 147, 4 (2025), 041710.

[30] Richard S Sutton. 1988. Learning to predict by the methods of temporal differences. *Machine learning* 3, 1 (1988), 9–44.

[31] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12 (1999).

[32] Qwen Team. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[33] Xilin Wang, Jia Zheng, Yuanchao Hu, Hao Zhu, Qian Yu, and Zihan Zhou. 2025. From 2d cad drawings to 3d parametric models: A vision-language approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 7961–7969.

[34] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[35] Rundi Wu, Chang Xiao, and Changxi Zheng. 2021. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF international conference on computer vision*. 6772–6782.

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

[37] Haiyang Yu, Xiaocong Wang, Bin Li, and Xiangyang Xue. 2023. Chinese Text Recognition with A Pre-Trained CLIP-Like Model Through Image-IDS Aligning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 11909–11918. doi:10.1109/ICCV51070.2023.01097

[38] Haiyang Yu, Siyang Yi, Ke Niu, Minghan Zhuo, and Bin Li. 2025. UMIT: Unifying Medical Imaging Tasks via Vision-Language Models. arXiv:2503.15892 [cs.CV] https://arxiv.org/abs/2503.15892

[39] Haiyang Yu, Mengyang Zhao, Jinghui Lu, Ke Niu, Yanjie Wang, Weijie Yin, Weitao Jia, Teng Fu, Yang Liu, Jun Liu, and Hong Chen. 2025. EVE: Towards End-to-End Video Subtitle Extraction with Vision-Language Models. arXiv:2503.04058 [cs.CV] https://arxiv.org/abs/2503.04058

[40] Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, Zhangwei Gao, Erfei Cui, Xuehui Wang, Yue Cao, Yangzhou Liu, Xingguang Wei, Hongjie Zhang, Haomin Wang, Weiye Xu, Hao Li, Jiahao Wang, Nianchen Deng, Songze Li, Yinan He, Tan Jiang, Jiapeng Luo, Yi Wang, Conghui He, Botian Shi, Xingcheng Zhang, Wenqi Shao, Junjun He, Yingtong Xiong, Wenwen Qu, Peng Sun, Penglong Jiao, Han Lv, Lijun Wu, Kaipeng Zhang, Huipeng Deng, Jiaye Ge, Kai Chen, Limin Wang, Min Dou, Lewei Lu, Xizhou Zhu, Tong Lu, Dahua Lin, Yu Qiao, Jifeng Dai, and Wenhai Wang. 2025. InternVL3: Exploring Advanced Training and Test-Time Recipes for Open-Source Multimodal Models. arXiv:2504.10479 [cs.CV] https://arxiv.org/abs/2504.10479