CRMoD-CAD: Unified Multi-Language CAD Code Generation with Concordant Reward

Anonymous ACL submission

Abstract

We present CRMoD-CAD, a unified framework that generates CAD code in OpenSCAD, CadQuery, and JSCAD from natural language. Training a single model across languages is hard: we identify a seesaw effect where improving one language degrades others in later stages, and reward hacking where models exploit partial IoU by generating oversimplified geometries. We solve both problems. Multi-Expert On-Policy Distillation provides token-level supervision for fine-grained per-language adjustment, significantly outperforming naive mixed-language RL. Concordant Reward RL combines IoU with a rubric-based VLM judge to suppress reward hacking on hard samples. Our key insight: all CAD languages compile to STL meshes, making geometric rewards language-agnostic and enabling RL to scale across languages at zero marginal cost. We construct UniCAD, a multi-language dataset with diverse difficulty levels sourced from existing datasets and open-source community designs. Experiments show CRMoD-CAD outperforms leading open-source and closed-source models on multi-language CAD code generation. Model weights and inference examples are available at `https://anonymous.4open.science/r/CRMoD-CAD-Emnlp-Example-FC2C`.

1 Introduction

Computer-Aided Design (CAD) code generation from natural language has garnered increasing attention as a means to democratize parametric 3D modeling (Jayaraman et al., 2022; Wu et al., 2021; Khan et al., 2024). However, real-world CAD ecosystems span multiple languages. Mechanical engineers rely on CadQuery for its parametric Python API, hobbyists prefer OpenSCAD (OpenSCAD Contributors, 2024) for its declarative CSG syntax, and web platforms adopt JSCAD for browser-native deployment. Requiring a separate model per language multiplies deployment costs and forgoes cross-language knowledge sharing. Yet existing methods uniformly target a single language (Niu et al., 2026; Guan et al., 2025; He et al., 2025; Doris et al., 2026), leaving the multilingual setting largely unexplored.

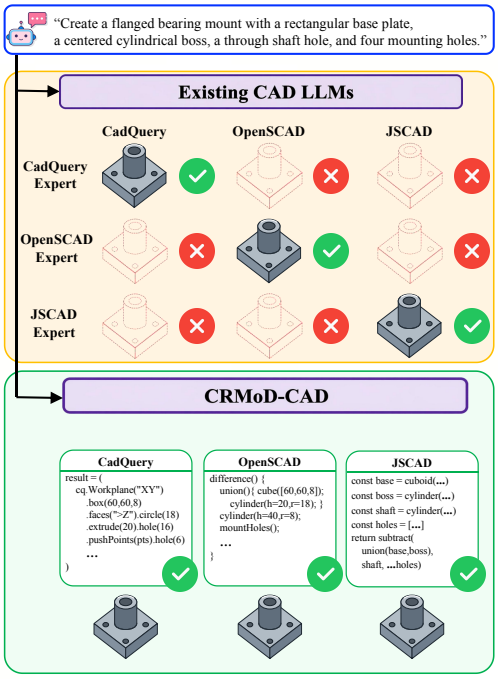

Figure 1: Overview of the CRMoD-CAD framework. Users provide text descriptions, and the system generates executable CAD code that compiles into precise 3D models.

A key observation makes multi-language CAD code generation tractable: *geometric rewards are language-agnostic*. All CAD languages compile to a shared output space (STL meshes), so IoU-based RL rewards transfer across languages at zero marginal cost without requiring additional annotation or evaluation infrastructure. This insight motivates us to construct **UniCAD**, a multi-language dataset spanning OpenSCAD, CadQuery, and JSCAD. Existing datasets predominantly contain easy

samples from programmatic templates (Khan et al., 2024; Alam and Ahmed, 2024), creating a difficulty ceiling that limits model capability. We additionally source hard samples from open-source CAD communities where practitioners share complex parametric designs. These community samples span a broader difficulty spectrum and better align with real-world user needs, reflecting the objects people actually want to create rather than synthetic templates.

While language-agnostic rewards enable multi-language RL at negligible cost, naively training a single model on mixed-language data introduces challenges. We empirically observe a *seesaw effect*: during joint RL training, improving one CAD language's generation capability consistently degrades others in later stages. The IoU reward provides only a scalar score per rollout, insufficient for the model to disentangle and independently optimize language-specific generation skills. Cross-language gradient interference causes the model to oscillate between languages rather than converge for all simultaneously. We address this with on-policy distillation (OPD) (Song and Zheng, 2026), which naturally provides token-level supervision. We train language-specific experts separately using RL, then distill them into a unified student via on-policy knowledge transfer (Zhao et al., 2026; Yang et al., 2026): the student generates its own rollouts while each expert provides fine-grained token-level guidance on its respective language. This **Multi-Expert On-Policy Distillation (M-OPD)** enables precise per-language adjustment without cross-language interference, resolving the seesaw effect.

Beyond the training paradigm, we identify a challenge in the reward design itself. On hard samples with complex geometry, models exploit partial volumetric overlap by generating oversimplified shapes. A prompt like "create a gear with 12 teeth" produces a simple cylinder achieving high IoU without capturing any geometric detail. Prior efforts to address reward reliability remain insufficient. CAD-RL (Niu et al., 2026) introduces LLM-based code evaluation, but current LLMs have weak CAD code comprehension and lack fine-grained scoring dimensions for reliably judging design intent. CADFusion (Wang et al., 2025a) adopts visual feedback from VLMs, but without structured rubrics, standalone VLM judges introduce substantial noise. We design **Concordant Reward RL**: combining IoU with a *rubric-based* visual language model (VLM) judge that evaluates rendered image pairs along explicit scoring dimensions. Rubrics as Rewards (Gunjal et al., 2025) demonstrates that using rubric-based structured feedback as on-policy RL rewards significantly outperforms direct Likert-based LLM judge scoring, while yielding better alignment for smaller judges and reducing performance variance across judge scales. We adapt this principle to the visual CAD domain, providing a structured rubric with five orthogonal dimensions that guide the VLM judge to score independently along explicit geometric criteria. Crucially, the judge score is not directly incorporated into the reward. Instead, it ranks rollouts, and a concordance bonus applies to the raw IoU reward only when IoU ranking and judge ranking agree, avoiding multi-dimensional reward interference while suppressing reward hacking.

We propose **CRMoD-CAD**, a comprehensive framework for unified multi-language CAD code generation. Our main contributions are:

- We construct UniCAD, a multi-language CAD dataset covering OpenSCAD, CadQuery, and JSCAD with easy samples from GenCAD-Code and hard samples from open-source community designs, built through reverse generation and round-trip verification, with 1k SFT samples per language equipped with synthetic chain-of-thought reasoning.
- We propose M-OPD to resolve the seesaw effect in mixed-language RL by training language-specific experts separately and distilling them into a unified model via token-level supervision, enabling fine-grained per-language generation adjustment.
- We propose Concordant Reward RL, introducing a rubric-based VLM judge as an auxiliary ranking signal to complement IoU. The judge score ranks rollouts, and a concordance bonus applies only when both rankings agree, amplifying rewards for rollouts excelling in both appearance and geometry while reducing gradient interference and suppressing reward hacking.

2 Related Work

2.1 CAD Code Generation

CAD code generation has evolved from command-sequence prediction to code-level parametric mod-

eling. DeepCAD (Wu et al., 2021) and SolidGen (Jayaraman et al., 2022) pioneer autoregressive generation of CAD command sequences. Subsequent methods leverage vision-language models: CAD-MLLM (Xu et al., 2024) and GenCAD (Alam and Ahmed, 2024) map image inputs to CAD commands, Img2CAD (You et al., 2025) separates structure prediction from parameter regression, and CAD2Program (Wang et al., 2025b) introduces flexible code representations for richer expressiveness. A growing body of work adopts CadQuery as the target language (He et al., 2025; Doris et al., 2026; Guan et al., 2025), with CAD-RL (Niu et al., 2026) integrating CoT reasoning with RL post-training, CADCodeVerify (Alrashedy et al., 2025) introducing VLM-based iterative verification, and ToolCAD (Gong et al., 2026) treating LLMs as tool-using agents. Despite this rapid progress, all existing methods target a single language. Our work is the first to pursue a unified model serving multiple CAD scripting languages simultaneously, exploiting their shared geometric output space to enable efficient cross-language training.

2.2 RL for Code Generation

Reinforcement learning (Rafailov et al., 2023; Shao et al., 2024) has been increasingly applied to code generation beyond supervised fine-tuning (Jiang et al., 2026). CodeRL (Le et al., 2022) and LEDEX (Jiang et al., 2024) use PPO to maximize execution-based rewards. Hybrid approaches such as StepCoder (Dou et al., 2024) and Swe-RL (Wei et al., 2026) combine supervised and reward-driven objectives. In the CAD domain specifically, CAD-RL (Niu et al., 2026) introduces LLM-based code evaluation as a reward signal, but current LLMs have weak CAD code comprehension and lack fine-grained scoring dimensions for reliably judging design intent. CADFusion (Wang et al., 2025a) adopts visual feedback from VLMs through DPO, but without structured rubrics, standalone VLM judges introduce substantial noise. ReCAD (Li et al., 2026) demonstrates hierarchical RL for complex CAD operations, and cadrille (Kolodiazhnyi et al., 2025) shows the effectiveness of online RL for multimodal CAD reconstruction. A parallel line of work explores structured reward signals: RaR (Gunjal et al., 2025) introduces an on-policy RL method that extends RLVR beyond verifiable domains by using rubric-based feedback as reward signals, significantly outperforming Likert-based LLM judge baselines while yielding better alignment for smaller judges and reducing variance across judge scales. We build on this insight by adapting rubric-based evaluation to the visual CAD domain, where rubrics guide a VLM judge to assess rendered 3D models along explicit geometric dimensions. However, unlike prior work that uses rubric scores as standalone rewards, we additionally combine them with IoU through a concordance mechanism that ensures multi-dimensional reward consistency.

2.3 OPD for Language Models

Offline knowledge distillation (Gu et al., 2024) trains a student on static teacher outputs, suffering from distribution mismatch as the student's generation behavior evolves during training. OPD (Song and Zheng, 2026) addresses this by having the student generate its own rollouts while the teacher provides token-level guidance on these trajectories, ensuring training signals remain aligned with the student's current policy. OPSD (Zhao et al., 2026) demonstrates a self-distillation variant where teacher and student share parameters but differ in input access, achieving high sample efficiency without a separate teacher model. GOPD (Yang et al., 2026) extends the framework with reward extrapolation, enabling the student to surpass the teacher. More recently, GLM-5 (Zeng et al., 2026) applies M-OPD at scale, merging domain-specific RL experts back into a unified foundation model for deployment. M-OPD can effectively consolidate diverse expert capabilities into a single model. We therefore adapt it to the *multi-language* CAD code generation setting, where language-specific experts specialize in different programming languages that share a common output space, and the goal is to unify their generation capabilities into a single deployment-efficient model.

3 UniCAD: Multi-Language Dataset

We construct the UniCAD dataset from two complementary sources to cover different difficulty levels, and derive SFT and RL training data for multi-language CAD code generation, as illustrated in Figure 2. Easy samples come from GenCAD-Code (Doris et al., 2026), which provides high-quality CadQuery code with relatively simple geometric compositions. Hard samples come from open-source communities, featuring complex structures that better reflect real-world design needs. We observe a key asymmetry: *models are signif-*

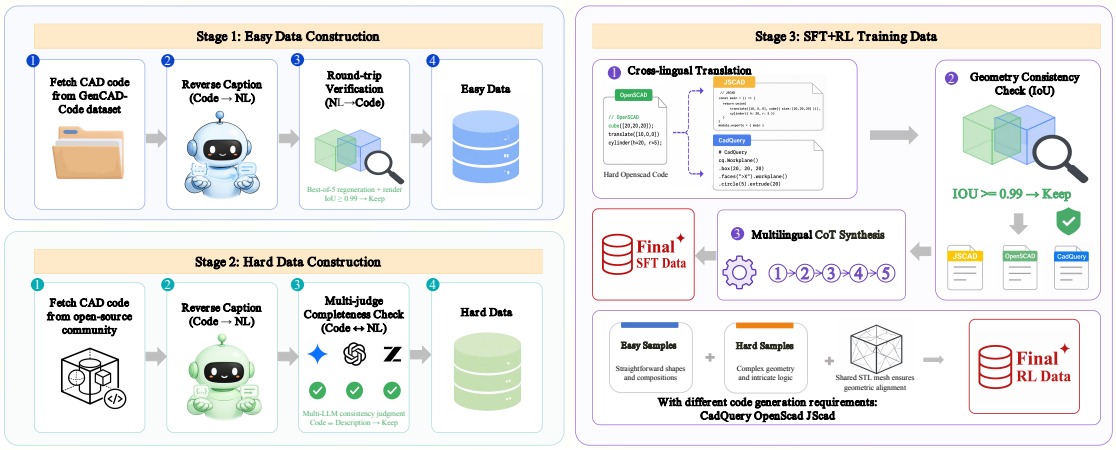

Figure 2: UniCAD data construction pipeline. Easy samples are obtained via reverse captioning and round-trip IoU verification, while hard samples from open-source communities are validated through three-judge unanimous agreement. Both splits are used to construct SFT and RL data with a language-agnostic geometric IoU reward.

icantly better at generating detailed descriptions from code than generating code from descriptions. We exploit this reverse direction to construct high-quality natural language descriptions. From verified samples, we derive 3k SFT instances with high-quality CAD design chain-of-thought reasoning synthesized by GPT-5.2, and a 21k RL training set using language-agnostic geometric IoU rewards.

3.1 Data Sources and Difficulty Tiers

Easy Samples from GenCAD-Code. We source easy samples from GenCAD-Code (Doris et al., 2026), which provides CadQuery programs with basic primitives and boolean operations. For each program, GPT-5.2 generates a natural language description via reverse generation. We verify quality through round-trip validation: GPT-5.2 regenerates code from the description with best-of-5 sampling, retaining only samples where at least one attempt achieves IoU $\geq$ 0.99 against the original. This yields 14k validated easy samples for RL training across all three languages.

Hard Samples from Open-Source Communities. We collect human-crafted CAD code from open-source communities, featuring advanced structures with loops, modules, conditional geometry, and multi-part assemblies. GPT-5.2 generates detailed descriptions via reverse generation. Since these designs are too complex for reliable round-trip IoU verification, we employ multi-judge voting: three independent frontier models (Gemini-3-Pro (Google DeepMind, 2025), GPT-5.2, and GLM-5) assess whether each description completely captures the geometric and parametric information in the source code. A hard sample is retained only upon unanimous agreement from all three judges, yielding 8k validated samples. Dataset statistics and filtering details are in Section C.

3.2 SFT and RL Data Construction

SFT Data. Since the open-source community designs are predominantly in OpenSCAD, multi-language code conversion is only required for SFT data and is performed exclusively on the hard community samples. We employ GPT-5.2 to convert the hard samples into CadQuery and JSCAD, retaining only conversions where the rendered 3D objects achieve IoU $\geq$ 0.99 before and after translation. From these verified conversions, we select 1k representative samples per language, 3k total, for supervised fine-tuning. For each of these hard SFT samples, GPT-5.2 further generates high-quality CoT reasoning that decomposes the complex geometric specification into step-by-step construction plans. Thus all CoT data in UniCAD is synthesized from the hard community samples. The synthesis prompt and examples are provided in Section L.

RL Data. For RL training, multi-language code conversion is unnecessary. The remaining 7k hard samples and all 14k easy samples from GenCAD-Code constitute the RL training set of 21k samples. Since any CAD language can be compiled to STL meshes for IoU computation, a single-language

source code suffices as the reward signal for training across all three target languages. The reward computation is performed entirely in the language-agnostic geometric output space.

4 Proposed Method

4.1 Overview

Given a natural language description of a 3D object, CRMoD-CAD generates executable CAD code in any of three target languages: OpenSCAD, CadQuery, or JSCAD. Our training pipeline proceeds in three stages: supervised fine-tuning on UniCAD with synthetic CoT reasoning, language-specific expert training using Concordant Reward RL (Section 4.3) that combines IoU with a rubric-driven VLM judge through a concordance mechanism, and M-OPD (Section 4.4) that consolidates the three experts into a single unified model via on-policy knowledge transfer.

4.2 Supervised Fine-Tuning

We first train three language-specific expert models via supervised fine-tuning on the UniCAD SFT set. Each expert is initialized from Qwen3.5-9B and fine-tuned on 1k samples of its target language. The training targets include synthetic CoT reasoning traces that decompose complex geometric specifications into step-by-step construction plans before the code output. The SFT objective minimizes the standard next-token prediction loss

$$\mathcal{L}_{\text{SFT}} = -\sum_{t=1}^{T} \log p_\theta(y_t \mid x, y_{<t}). \qquad (1)$$

where x is the input description and $y = (y_1, \ldots, y_T)$ is the target sequence of reasoning trace followed by code. This stage establishes proper reasoning style and format conventions in each language for subsequent RL training.

4.3 Concordant Reward RL

After supervised fine-tuning, we apply reinforcement learning to improve generation quality. Relying on IoU as the sole reward leads to reward hacking on hard samples, while existing alternatives based on code-level LLM evaluation or unstructured visual feedback introduce their own limitations as discussed in Section 2. Our Concordant Reward RL addresses this by combining IoU with a rubric-based VLM judge, using a concordance mechanism to ensure multi-dimensional reward consistency.

IoU Reward. We retain IoU as the hard geometric reward. Given generated code c and ground-truth code c^*, we render both to 3D models and compute:

$$R_{\text{iou}} = \frac{|\text{Render}(c) \cap \text{Render}(c^*)|}{|\text{Render}(c) \cup \text{Render}(c^*)|}. \qquad (2)$$

VLM Judge. Rather than judging generated code directly, a task at which current models are weak, we leverage VLMs' strong visual comparison capabilities. We render both the generated model and the reference model into images, then prompt a VLM to assess structural consistency based on a structured rubric (Gunjal et al., 2025). The rubric decomposes evaluation into five orthogonal dimensions: outline & proportion, structural completeness, cutout & detail consistency, layout & symmetry, and topology & connectivity. Each dimension scores independently as 0, 1, or 2. The total judge score sums all five dimensions:

$$R_{\text{judge}} = \sum_{k=1}^{5} S_k\big(I(c),\, I(c^*)\big) \;\in\; \{0, \ldots, 10\}, \qquad (3)$$

where S_k is the score for dimension k, and $I(\cdot)$ renders code into an image for visual comparison.

Concordance Bonus. Rollouts scoring highly on *both* IoU and the visual judge represent samples that excel in both internal geometry and external appearance. Directly combining rewards from different dimensions introduces optimization interference: IoU and visual quality operate on different scales and produce conflicting gradient directions. Instead, we use the judge score solely to rank rollouts and selectively amplify the reward where both rankings agree. Rollouts with inconsistent rankings retain the raw IoU score as the primary reward. The judge serves as an auxiliary signal that amplifies rewards for rollouts excelling in both dimensions, helping the model learn superior code design patterns while reducing gradient interference and suppressing IoU reward hacking-cases where high geometric overlap comes from oversimplified shapes lacking visual fidelity.

Concretely, we rank the G rollouts independently by IoU and by judge score. Starting from rank 1, if the top-ranked rollout by IoU is also top-ranked by the judge, it receives a bonus α_1 added directly to its raw IoU score. If the second-ranked rollout also agrees on both metrics, it receives α_2. This cascading process halts as soon as the rankings

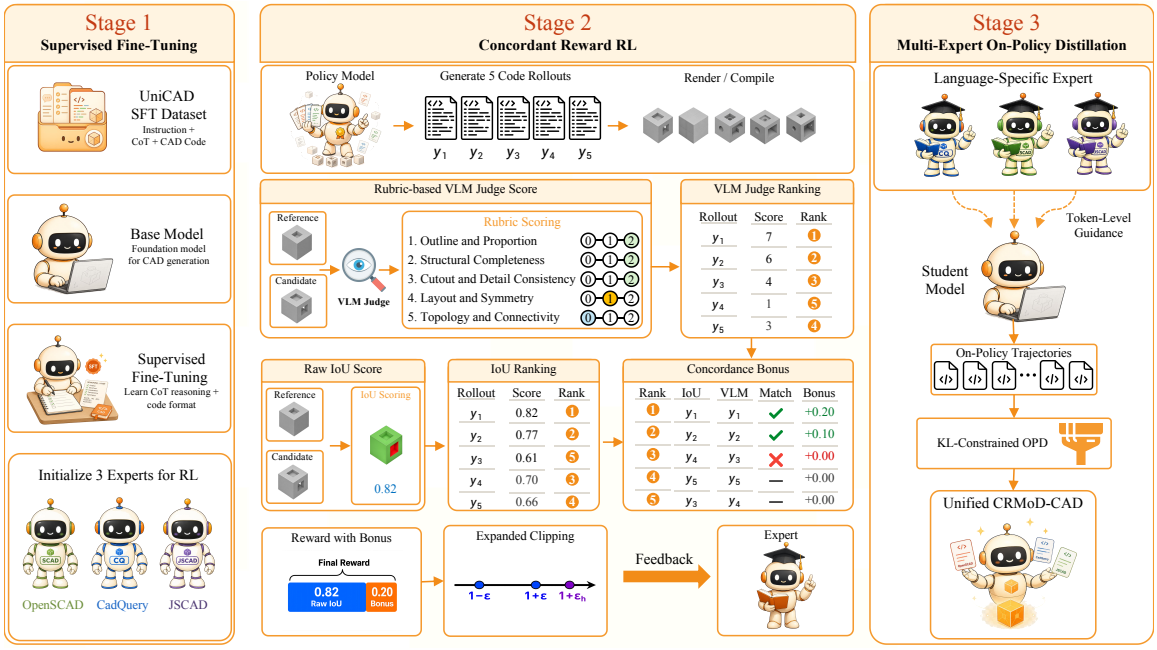

Figure 3: Architecture of CRMoD-CAD. Left: UniCAD data construction and CoT-based SFT. Middle: Concordant Reward RL combining IoU with VLM judge. Right: M-OPD to unified model.

diverge at any level:

$$B^{(i)} = \sum_{k=1}^{G} \alpha_k \cdot \mathbf{1}\left[\mathrm{rk}_{\mathrm{iou}}(i) = \mathrm{rk}_{\mathrm{judge}}(i) = k\right] \cdot \prod_{j=1}^{k-1} C_j, \tag{4}$$

where $B^{(i)}$ is the concordance bonus for rollout i, $\mathrm{rk}_{\mathrm{iou}}(i)$ and $\mathrm{rk}_{\mathrm{judge}}(i)$ denote its rank by IoU and judge score respectively, $\sigma(j)$ denotes the rollout ranked j-th by IoU, and $C_j = \mathbf{1}\left[\mathrm{rk}_{\mathrm{judge}}(\sigma(j)) = j\right]$ is the concordance indicator at rank j, ensuring that the IoU-top-j rollout is also judge-top-j. In practice, we set $\alpha_k = 0.2/2^{k-1}$, yielding $0.2, 0.1, 0.05, \ldots$, halving the bonus at each successive rank.

Group Normalization. After adding the concordance bonus to raw IoU scores, we normalize the combined reward within the rollout group following standard GRPO practice:

$$\tilde{R}^{(i)} = \frac{\left(R_{\mathrm{iou}}^{(i)} + B^{(i)}\right) - \mu}{\sigma + \epsilon}, \tag{5}$$

where μ and σ are the mean and standard deviation computed over the group $\{R_{\mathrm{iou}}^{(i)} + B^{(i)}\}_{i=1}^{G}$, and ϵ is a small constant for numerical stability.

Expanded PPO Clipping. Standard PPO (Shao et al., 2024) clips the importance ratio within $[1-\epsilon, 1+\epsilon]$. For rollouts that received a concordance bonus, we expand the upper clip bound to allow the policy to make larger updates toward these high-confidence solutions:

$$\mathcal{L}_{\mathrm{PPO}} = \mathbb{E}\left[\min\left(r_t \hat{A}_t,\ \mathrm{clip}(r_t, 1-\epsilon, 1+\epsilon_h^{(i)})\hat{A}_t\right)\right], \tag{6}$$

where $r_t = \pi_\theta(y_t|x, y_{<t})/\pi_{\mathrm{old}}(y_t|x, y_{<t})$ is the importance ratio, $\hat{A}_t$ is the estimated advantage, and $\epsilon_h^{(i)} = \epsilon_{\mathrm{high}}$ if $B^{(i)} > 0$ and $\epsilon_h^{(i)} = \epsilon$ otherwise. This selective expansion ensures that only rollouts validated by both IoU and visual quality receive enhanced gradient signal, while standard clipping applies to all other samples.

4.4 Multi-Expert On-Policy Distillation

Naïvely training a single model via mixed-language RL leads to cross-language gradient interference in later stages, causing the unified model to significantly underperform single-language experts. Traditional offline distillation (Gu et al., 2024) suffers from distribution mismatch because static teacher data does not reflect the student's evolving generation behavior.

We address both issues with M-OPD, proceeding in two stages.

Stage 1: Language-Specific Expert Training. We train three separate expert models $\{\pi_E^{\mathrm{scad}}, \pi_E^{\mathrm{cq}}, \pi_E^{\mathrm{jcad}}\}$ using Concordant Reward RL on their respective language datasets, each optimizing exclusively on one CAD language to achieve peak performance without cross-language interference.

Stage 2: OPD to Unified Model. We distill all three experts into a single unified student π_θ. At each iteration, the student generates its own rollout for a prompt x with target language l, and the corresponding expert π_E^l provides token-level distributional guidance. We formulate the objective as dense KL-constrained RL (Yang et al., 2026):

$$J_{\text{OPD}}(\theta) = \max_{\theta} \mathbb{E}_{\substack{x\sim\mathcal{D}_l \\ y\sim\pi_\theta}}\Big[\lambda \log \frac{\pi_E^l(y|x)}{\pi_{\text{ref}}(y|x)} - D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\Big], \tag{7}$$

where π_{ref} is the student's SFT checkpoint, $\log \frac{\pi_E^l(y|x)}{\pi_{\text{ref}}(y|x)}$ serves as a dense token-level reward signal, and $\lambda = 1.25$ enables reward extrapolation that encourages the student to surpass individual experts. The unified student surpasses the OpenSCAD expert and retains 96–99% of CadQuery and JSCAD expert performance, outperforming the mixed-language RL baseline by +2.5% average IoU across all three languages.

5 Experiments

5.1 Implementation Details

We use Qwen3.5-9B as the base model throughout the entire pipeline on 8 NVIDIA H20 GPUs. The VLM judge uses Doubao-Seed-2.0 (ByteDance Seed Team, 2026) with a structured rubric containing five orthogonal scoring dimensions. We use the AdamW optimizer with cosine annealing, learning rate 1e-5 for SFT and 1e-6 for RL. The distillation reward scaling factor is $\lambda = 1.25$. Full hyperparameters and training times are provided in Section B.

5.2 Datasets and Metrics

Training Data. The SFT stage uses 3k samples (1k per language) with CoT reasoning synthesized by GPT-5.2, all sourced from hard community samples. The RL stage uses 21k samples in total, comprising 14k easy samples from GenCAD-Code and 7k hard samples from open-source communities (after reserving 1k hard for SFT).

Evaluation. We evaluate on the UniCAD test set, which contains 200 easy samples and 200 hard samples, all excluded from both SFT and RL training data. Each difficulty tier is tested independently on all three CAD languages. We adopt two primary metrics. **Intersection-over-Union (IoU)** evaluates volumetric overlap between predicted and reference shapes. **Executability** directly reflects the functional validity of the generated code.

5.3 Main Results

Table 1 presents the main results. We compare CRMoD-CAD against five pretrained LLMs and the Qwen3.5-9B base model, while leaving language-specific expert ablations to Table 2.

CRMoD-CAD achieves the best average IoU across all three languages, 80.0% on easy samples and 40.4% on hard samples, surpassing GPT-5.2 by +6.6% on easy and +20.1% on hard. On OpenSCAD, the unified model reaches 83.2% IoU on easy and 51.2% on hard, demonstrating beneficial cross-language transfer. The advantage on hard community samples is particularly pronounced: per-language hard IoU reaches 51.2/38.4/31.6 on OpenSCAD/CadQuery/JSCAD, nearly doubling GPT-5.2's 27.1/20.1/13.8, while all open-source baselines and even GLM-5 stay below 25% on every language. Executability follows the same trend, with CRMoD-CAD attaining 99.32% on easy and 96.4% on hard for OpenSCAD, well ahead of GPT-5.2's 91.28/75.5, confirming gains hold under both geometric fidelity and code-level validity. This confirms that the concordant reward—by boosting rollouts that rank highly under both IoU and visual quality—yields substantial gains on geometrically complex prompts, where naive rewards tend to collapse into oversimplified shapes. Notably, frontier models such as GLM-5 and GPT-5.2 already achieve strong zero-shot performance on easy CAD generation, yet our 9B fine-tuned model still outperforms them by a clear margin across all languages and difficulty levels, validating language-specific expert training followed by OPD merging.

5.4 Multi-Language CAD Generation

Table 2 compares different strategies for multi-language CAD code generation.

Single-language training causes catastrophic forgetting. Training exclusively on one CAD language destroys performance on others. The OpenSCAD expert achieves 82.4% IoU on OpenSCAD but drops CadQuery to 4.8% and JSCAD to 14.2%, far below the untrained base model. This pattern holds across all three experts, confirming that specialized RL training overwrites previously learned language capabilities.

Mix RL suffers from cross-language interference. Joint training on all three languages reaches 81.5%/78.1%/72.8% respectively. While it avoids catastrophic forgetting, each language underperforms its corresponding single-language expert due

Table 1: Comparison on the UniCAD dataset. The upper block shows pretrained LLMs without fine-tuning and the lower block shows fine-tuned models. Each cell reports **easy/hard** IoU or executability. Bold indicates the best and underline the second-best.

Model	OpenSCAD		CadQuery		JSCAD		Avg. IoU(%)↑	Avg. Exec.(%)↑
	IoU(%)↑	Exec.(%)↑	IoU(%)↑	Exec.(%)↑	IoU(%)↑	Exec.(%)↑		
Gemini 2.5 Pro (Comanici et al., 2025)	54.6/12.4	72.76/56.5	45.2/11.4	64.89/25.3	37.0/12.0	56.23/27.1	45.6/11.9	64.6/36.3
GPT-4o (OpenAI, 2024)	61.8/18.0	82.67/68.3	50.3/14.9	72.64/29.6	40.8/12.7	62.64/31.8	51.0/15.2	72.7/43.2
GLM-5 (Zeng et al., 2026)	75.3/24.8	89.12/73.4	64.8/15.2	75.51/30.5	70.4/16.1	84.35/43.0	70.2/18.7	83.0/49.0
GPT-5.2-high (OpenAI, 2025)	74.7/27.1	91.28/75.5	73.8/20.1	89.76/36.5	71.7/13.8	88.21/45.5	73.4/20.3	89.8/52.5
Qwen3.5-9B (Qwen Team, 2026)	52.8/12.5	68.43/54.8	21.1/4.7	35.72/13.5	25.1/6.1	38.96/17.0	33.0/7.8	47.7/28.4
CRMoD-CAD	**83.2/51.2**	**99.32/96.4**	**79.6/38.4**	**98.65/95.7**	**77.3/31.6**	**98.17/95.4**	**80.0/40.4**	**98.7/95.8**

Table 2: Multi-language CAD code generation comparison (IoU%). Single-language experts achieve strong performance on their target language but suffer catastrophic forgetting on others. M-OPD achieves the best overall performance across all three languages. All models use the Qwen3.5-9B base architecture trained with pure RL.

Method	SCAD	CQ	JSCAD	Avg.
Base Model	52.8	21.1	25.1	33.0
SCAD Expert	82.4	4.8	14.2	33.8
CQ Expert	43.5	**80.4**	33.7	52.5
JSCAD Expert	41.6	3.2	**79.2**	41.3
Mix RL	81.5	78.1	72.8	77.5
CRMoD-CAD	**83.2**	79.6	77.3	**80.0**

Table 3: Ablation study of CRMoD-CAD on the UniCAD dataset (OpenSCAD). We progressively add components to study their individual contributions. ↑ indicates higher is better.

Configuration	IoU(%)↑	Exec.(%)↑
SFT only	58.5	90.64
+ IoU RL	79.6	98.04
+ Visual Judge (sum norm)	79.2	97.86
+ Concordance Bonus	81.6	98.40
+ Expanded Clip (Full)	**82.4**	**98.74**

to conflicting gradient signals across languages.

M-OPD outperforms Mix RL on all three languages. Our M-OPD achieves 83.2%/79.6%/77.3%, surpassing Mix RL by +1.7%/+1.5%/+4.5% on OpenSCAD/CadQuery/JSCAD respectively, with an average improvement of +2.5% IoU. Notably, the unified model even exceeds the OpenSCAD single-expert by +0.8%, suggesting that cross-language representation sharing through the unified backbone provides beneficial geometric reasoning transfer.

5.5 Ablation Study

We progressively incorporate each component of our concordant reward design as shown in Table 3. SFT alone achieves 90.64% executability but only 58.5% IoU, indicating correct syntax acquisition without accurate geometric reconstruction.

Introducing IoU-based RL raises IoU to 79.6%. However, on complex samples the model exploits partial volumetric overlap by generating oversimplified geometries. Naively normalizing and summing IoU and judge scores actually degrades IoU to 79.2% (-0.4%), as conflicting gradient directions harm geometric optimization.

Our concordance bonus yields 81.6% IoU (+2.4%) by using the judge score solely for ranking and amplifying rollouts where both IoU and visual quality rank highly, effectively suppressing reward hacking. Finally, expanding the PPO clip upper bound adds +0.8% reaching 82.4%, enabling larger policy updates toward high-reward solutions.

6 Conclusion

We presented CRMoD-CAD, a framework for unified multi-language CAD code generation. UniCAD provides a multi-language dataset constructed through reverse generation and round-trip verification. Concordant Reward RL combines IoU with a rubric-based VLM judge through a concordance mechanism, amplifying reward signals from samples that excel in both geometry and visual fidelity. M-OPD trains language-specific experts and distills them into a unified model, resolving the seesaw effect with +2.5% average IoU over mixed-language RL. The unified model surpasses the OpenSCAD single-expert while retaining 96–99% of CadQuery and JSCAD expert capabilities. Future work may explore interactive refinement, multi-turn generation, and extension to additional CAD languages.

Limitations

Two limitations remain. First, our framework currently covers three CAD languages and has not been extended to others such as FreeCAD Python or SolidPython. Second, our VLM judge operates on a single rendered view. Multi-view scoring would be more comprehensive but multiplies per-rollout cost. A natural next step is to replace the judge with a smaller open-source model and then introduce multi-view scoring.

References

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*.

Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Haider Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. 2025. Generating cad code with vision-language models for 3d designs. In *International Conference on Learning Representations*, volume 2025, pages 52236–52262.

ByteDance Seed Team. 2026. Seed2.0 model card: Towards intelligence frontier for real-world complexity. Technical Report.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Anna C Doris, Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. 2026. Cad-coder: An open-source vision-language model for computer-aided design code generation. *Journal of Mechanical Design*, 148(7):071702.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, and 1 others. 2024. Stepcoder: improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4571–4585.

Yifei Gong, Xing Wu, Wenda Liu, and Kang Tu. 2026. Toolcad: Exploring tool-using large language models in text-to-cad generation with reinforcement learning. *arXiv preprint arXiv:2604.07960*.

Google DeepMind. 2025. Gemini 3 Pro model card. Technical Report.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2024. Minillm: Knowledge distillation of large language models. In *International Conference on Learning Representations*, volume 2024, pages 32694–32717.

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. In *Advances in Neural Information Processing Systems*, volume 38, pages 59765–59789.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*.

Changqi He, Shuhan Zhang, Liguo Zhang, and Jiajun Miao. 2025. Cad-coder: Text-guided cad files code generation. *arXiv preprint arXiv:2505.08686*.

Pradeep Kumar Jayaraman, Joseph G Lambourne, Nishkrit Desai, Karl DD Willis, Aditya Sanghi, and Nigel JW Morris. 2022. Solidgen: An autoregressive model for direct b-rep synthesis. *arXiv preprint arXiv:2203.13944*.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2026. A survey on large language models for code generation. *ACM Transactions on Software Engineering and Methodology*, 35(2):1–72.

Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya B Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. 2024. Ledex: Training llms to better self-debug and explain code. *Advances in Neural Information Processing Systems*, 37:35517–35543.

Mohammad S Khan, Sankalp Sinha, Talha U Sheikh, Didier Stricker, Sk A Ali, and Muhammad Z Afzal. 2024. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 37:7552–7579.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. 2025. cadrille: Multi-modal cad reconstruction with online reinforcement learning. *arXiv preprint arXiv:2505.22914*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Jiahao Li, Yusheng Luo, Yunzhong Lou, and Xiangdong Zhou. 2026. Recad: Reinforcement learning enhanced parametric cad model generation with vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 6190–6198.

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*, volume 40, pages 8160–8167.

OpenAI. 2024. GPT-4o system card. Technical Report.

OpenAI. 2025. Update to GPT-5 system card: GPT-5.2. Technical Report.

OpenSCAD Contributors. 2024. OpenSCAD: The programmer's solid 3D CAD modeller. https://openscad.org/.

Qwen Team. 2026. Qwen3.5: Towards native multimodal agents. Blog post.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Mingyang Song and Mao Zheng. 2026. A survey of on-policy distillation for large language models. *arXiv preprint arXiv:2604.00626*.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. 2025a. Text-to-cad generation through infusing visual feedback in large language models. *arXiv preprint arXiv:2501.19054*.

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

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida Wang. 2026. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *Advances in Neural Information Processing Systems*, 38:78500–78525.

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*, pages 6772–6782.

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*.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. 2026. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation. *arXiv preprint arXiv:2602.12125*.

Yang You, Mikaela Angelina Uy, Jiaqi Han, Rahul Thomas, Haotong Zhang, Yi Du, Hansheng Chen, Francis Engelmann, Suya You, and Leonidas Guibas. 2025. Img2cad: Reverse engineering 3d cad models from images through vlm-assisted conditional factorization. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*, pages 1–12.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, and 1 others. 2026. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. 2026. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*.

A Qualitative Results

Figure 4 presents generation results from our unified CRMoD-CAD model across all three CAD languages. For each example, we show the input text description alongside the rendered 3D model produced by the generated code. These results demonstrate that our model can generate geometrically accurate and visually faithful CAD code across diverse object categories and complexity levels.

B Implementation Details

All experiments are conducted on 8 NVIDIA H20 GPUs using Qwen3.5-9B as the base model.

SFT. We fine-tune on 3k samples (1k per language) with CoT reasoning synthesized by GPT-5.2. The learning rate is 1e-5 with cosine annealing. The maximum prompt length is 2048 tokens and maximum response length is 4096 tokens. Training takes approximately 24 hours.

Expert RL Training. Each language-specific expert is independently trained with a batch size of 128 and GRPO rollout count $G = 5$. We train for 120 steps on easy samples and 200 steps on hard samples. The learning rate is 1e-6 with cosine annealing. On 8 H20 GPUs, expert training on easy samples takes approximately 36 hours and on hard samples approximately 50 hours.

Mixed-Language RL. We use a total batch size of 384 with 128 samples per language per step, evenly drawn from OpenSCAD, CadQuery, and JSCAD. Training runs for 120 steps on easy samples.

M-OPD. M-OPD uses a total batch size of 384 for 120 steps. The distillation reward scaling factor is $\lambda = 1.25$.

VLM Judge. The judge uses Doubao-Seed-2.0 (ByteDance Seed Team, 2026) to assess structural consistency of rendered image pairs via a structured rubric with five orthogonal scoring dimensions.

Optimizer. We use the AdamW optimizer throughout all stages. The maximum prompt length is set to 2048 tokens and maximum response length to 4096 tokens.

C Dataset Construction Details

UniCAD covers three CAD languages with two difficulty tiers. All length statistics below are measured in tokens using the same tokenizer employed during training. **Easy samples**, 14k in total, are sourced from the GenCAD-Code dataset (Doris et al., 2026), providing CadQuery programs for standard geometric compositions. Easy samples are concise, with code averaging 497 tokens and prompts averaging 517 tokens. **Hard samples**, 8k in total, are collected from multiple open-source communities, featuring complex parametric designs with loops, modules, conditional geometry, and multi-part assemblies. Hard samples average 2,137 tokens in code and 729 tokens in prompt, roughly $4\times$ longer than easy samples. We cap code at 3,072 tokens and prompts at 1,536 tokens.

For data composition, 1k samples per language are selected from the hard samples for SFT, 3k total, each augmented with GPT-5.2-synthesized CoT reasoning. The remaining 7k hard samples plus all 14k easy samples form the RL training set of 21k samples. Since RL training computes IoU rewards directly from rendered STL meshes, multi-language code conversion is not required for the RL data.

For evaluation, we hold out 200 easy test samples and 200 hard test samples, all excluded from both SFT and RL training data. The test samples are evaluated independently on each of the three target languages.

D Reward Design Detailed Analysis

Table 4 presents a detailed comparison of three reward configurations evaluated on the 9B base model trained via M-OPD using hard samples. This experiment isolates the effect of our concordant reward design under our actual training setup.

Table 4: Reward design ablation on the 9B M-OPD model with hard samples. Simple summation of IoU and VLM judge degrades both IoU and judge score due to uncoordinated optimization, while our concordant reward with cascading bonus improves both metrics simultaneously.

Reward Configuration	IoU(%)	Judge(%)
IoU + Exec	38.3	52.9
IoU + Judge (sum norm) + Exec	37.5	51.2
Concordant (Ours)	**40.4**	**55.0**

The results reveal an important finding: IoU and visual quality optimize different aspects of code design: high IoU does not guarantee good appearance, and visually appealing outputs do not necessarily achieve high IoU. Naively adding the two signals through sum normalization actually *degrades* both

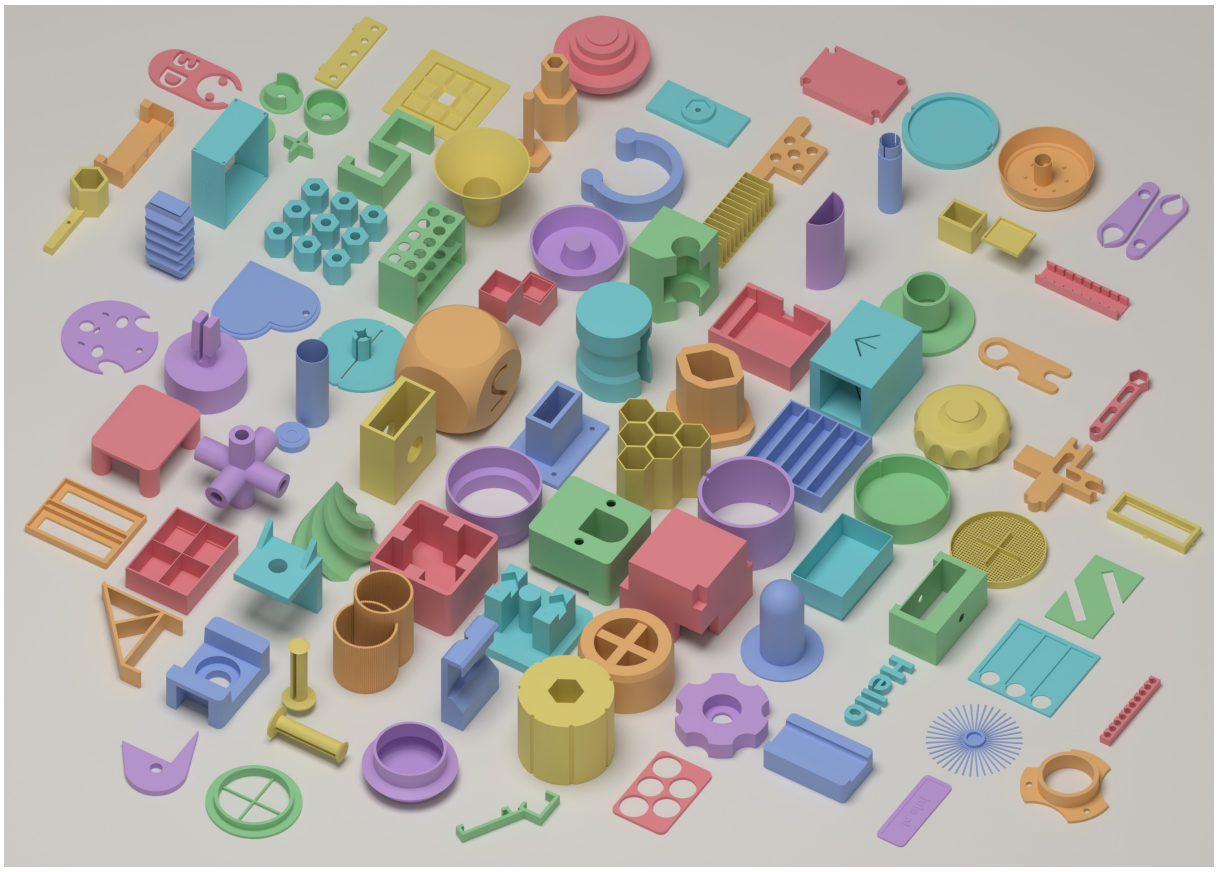

Figure 4: Generation results from the unified CRMoD-CAD model across CadQuery, OpenSCAD, and JSCAD, demonstrating its ability to generate complex CAD objects in multiple languages.

metrics, reducing IoU from 38.3% to 37.5% and judge score from 52.9% to 51.2%. This is because the oscillation between the two optimization directions introduces substantial learning noise during training, and direct summation damages both the hard, verifiable IoU reward signal and the visual quality signal simultaneously.

Our concordant reward addresses this from a different angle: rather than combining rewards additively, it grants a bonus only when the two signals agree in ranking, amplifying rewards for rollouts that excel in both dimensions. This allows the model to learn code design patterns that are simultaneously excellent in appearance and geometry, while reducing the gradient interference caused by naive multi-dimensional reward summation. This lifts IoU significantly from 38.3% to 40.4%, an improvement of +2.1%, while simultaneously improving judge score from 52.9% to 55.0%, a gain of +2.1%, validating that the concordance mechanism effectively filters noise while amplifying the learning signal from high-quality samples.

Concordance Bonus Magnitude. Table 5 analyzes the sensitivity of the concordance bonus magnitude α. All variants outperform the baseline (IoU + Exec), confirming that the concordance mechanism is robust across a range of bonus values. The optimal setting is $\alpha = 0.2$, which achieves the best trade-off between IoU and visual judge score. A smaller bonus ($\alpha = 0.1$) provides moderate gains in both metrics but under-exploits the concordance signal. A larger bonus ($\alpha = 0.4$) over-weights concordant rollouts, which slightly degrades IoU below $\alpha = 0.1$ as the stronger bonus amplifies a narrower subset of training samples, reducing effective batch diversity. Interestingly, $\alpha = 0.4$ still achieves a slightly higher judge score than $\alpha = 0.1$, suggesting that the visual quality dimension benefits from stronger concordance emphasis even when geometric fidelity suffers.

E Effect of CoT SFT

To isolate the contribution of the synthesized CoT SFT stage, we train each language expert directly from the base model via RL without the preceding SFT initialization. All other hyperparameters

Table 5: Sensitivity analysis of the concordance bonus magnitude α on the 9B M-OPD model with hard samples. All concordance variants outperform the baseline, with $\alpha = 0.2$ achieving the best overall performance.

Bonus α	**IoU(%)**	**Judge(%)**
Baseline (no bonus)	38.3	52.9
$\alpha = 0.1$	39.5	53.8
$\alpha = 0.2$ (Ours)	**40.4**	**55.0**
$\alpha = 0.4$	39.1	54.2

remain identical. Table 6 reports the results.

Table 6: Ablation on CoT SFT initialization. Each expert is evaluated on its target language. Removing the SFT stage consistently degrades IoU by 1–2% across all languages and difficulty levels, confirming that synthesized CoT reasoning provides a beneficial initialization for RL training.

Method	**SCAD**	**CQ**	**JSCAD**	**Avg.**
Easy Samples IoU%				
RL w/o SFT	80.6	78.8	77.5	79.0
SFT + RL (Ours)	**82.4**	**80.4**	**79.2**	**80.7**
Hard Samples IoU%				
RL w/o SFT	43.4	38.5	37.1	39.7
SFT + RL (Ours)	**45.1**	**40.3**	**38.7**	**41.4**

Removing SFT consistently degrades target-language IoU by 1.6–1.8% on easy samples and 1.6–1.8% on hard samples. The gap is remarkably uniform across all three languages, indicating that the benefit of CoT SFT is not language-specific but rather stems from establishing a structured reasoning style that generalizes across CAD languages. On hard samples the effect is slightly more pronounced, as the complex geometric specifications benefit more from the step-by-step decomposition learned during SFT. These results confirm that the GPT-5.2-synthesized CoT reasoning provides a meaningful initialization for subsequent RL training, helping the model adopt effective code generation reasoning patterns and output format conventions before reward-driven optimization begins.

F Cross-Language Forgetting Analysis

We provide detailed analysis of the catastrophic forgetting phenomenon observed in single-language expert training. As shown in Table 2, training the base model exclusively on one CAD language rapidly erodes capabilities on others:

- **OpenSCAD Expert**: Reaches 82.4% on OpenSCAD but drops CadQuery from 21.1% to 4.8%, a 77.3% relative decrease, and JSCAD from 25.1% to 14.2%, a 43.4% relative decrease.
- **CadQuery Expert**: Achieves 80.4% on CadQuery but degrades OpenSCAD from 52.8% to 43.5%, a 17.6% relative decrease. JSCAD remains relatively stable at 33.7%, suggesting that CadQuery training transfers more gracefully to JSCAD than to OpenSCAD.
- **JSCAD Expert**: Reaches 79.2% on JSCAD but OpenSCAD drops from 52.8% to 41.6%, a 21.2% relative decrease, and CadQuery drops from 21.1% to 3.2%, an 84.8% relative decrease.

This forgetting motivates our two-stage expert-then-distillation approach. M-OPD avoids this tradeoff by training experts independently on each language and then consolidating via OPD, achieving 83.2%/79.6%/77.3% across all three languages, retaining or even exceeding individual expert performance on their target languages while maintaining strong cross-language capabilities.

G Model Scale Extension

To verify that our framework naturally extends to stronger teachers, we train language-specific experts using Qwen3.5-27B and distill them into the same 9B student. Table 7 reports the results.

Table 7: Model scale extension experiment. 27B experts outperform 9B experts on every language, and the distilled 9B unified model obtains consistent gains, validating that our framework naturally benefits from larger teacher models without any architectural or procedural modification. Top: easy-sample IoU%. Bottom: hard-sample average IoU%.

Teacher	**Method**	**SCAD**	**CQ**	**JSCAD**	**Avg.**
Easy Samples IoU%					
27B	SCAD Expert	92.4	6.2	16.8	38.5
	CQ Expert	46.3	90.3	36.4	57.7
	JSCAD Expert	44.2	5.1	85.2	44.8
9B	M-OPD $\rightarrow$ 9B	83.2	79.6	77.3	80.0
27B	M-OPD $\rightarrow$ 9B	**90.1**	**88.7**	**81.3**	**86.7**
Hard Samples IoU%					
9B	M-OPD $\rightarrow$ 9B	44.2	39.1	37.9	40.4
27B	M-OPD $\rightarrow$ 9B	**49.5**	**43.8**	**41.7**	**45.0**

The 27B experts improve over their 9B counterparts by approximately 6–10% IoU on each language. After distillation into the same 9B student, the M-OPD unified model rises from 80.0%

to 86.7% average IoU on easy samples and from 40.4% to 45.0% on hard samples. OpenSCAD benefits most with a +6.9% gain, while JSCAD improves least at +4.0%, suggesting that languages with richer training signal from stronger teachers gain more from distillation. This confirms that stronger teachers provide higher-quality distributional supervision through OPD, and our framework benefits from teacher scaling without any modification to architecture or training procedure.

H Chamfer Distance Evaluation

To provide complementary geometric evaluation beyond IoU, we additionally report Chamfer Distance on both easy and hard test sets. Table 8 presents both mean CD and median CD across all three languages. Mean CD captures the average surface deviation while median CD is more robust to outliers caused by execution failures or degenerate geometries. Lower values indicate better surface alignment between generated and reference meshes.

On easy samples, the CD results are fully consistent with the IoU rankings reported in the main paper. CRMoD-CAD achieves the lowest mean CD across all three languages, with 4.73 on OpenSCAD, 6.82 on CadQuery, and 7.51 on JSCAD, substantially outperforming GPT-5.2 whose mean CD values are 7.95, 11.89, and 9.24 respectively. The median CD further confirms this advantage: CRMoD-CAD's median values are an order of magnitude lower than most baselines, indicating that the majority of generated shapes closely align with references rather than relying on a few easy samples to bring down the average.

On hard samples, all models exhibit significantly higher CD values due to the increased geometric complexity. CRMoD-CAD maintains its advantage with mean CD of 16.76 on OpenSCAD, 19.85 on CadQuery, and 21.34 on JSCAD, outperforming GPT-5.2 whose corresponding values are 17.86, 21.73, and 23.42. The gap between CRMoD-CAD and weaker baselines widens considerably on hard samples: GPT-4o's mean CD reaches 29.04 on OpenSCAD while Qwen3.5-9B reaches 37.61, nearly 2.2× the CD of CRMoD-CAD. Notably, the gap between mean and median CD is much larger for weaker models on hard samples, reflecting their higher variance and more frequent generation failures when facing complex parametric designs.

I VLM Judge Prompt

The VLM judge receives rendered images of both the generated model and the reference model, and assesses structural consistency between them. The prompt defines five orthogonal scoring dimensions: outline and proportion, structural completeness, cutout and detail consistency, layout and symmetry, and topology and connectivity. Each dimension is scored independently as 0, 1, or 2, and the total score is the sum of all five dimensions, ranging from 0 to 10. The VLM outputs a score and a brief reason for each dimension separately, forming a structured JSON array. This total score is used solely for rollout ranking and is not directly incorporated into the reward value. The complete prompt template used with Doubao-Seed-2.0 is shown in Figure 5.

J Training System Prompts

Each CAD language uses a language-specific system prompt during both SFT and RL training to guide the model's output format. The three prompts share similar overall structure and task definition, differing only in the target code language. Figure 6 shows the complete prompt template for all three languages together with an example user query.

K Design Description Synthesis

For each CAD code sample in the dataset, we use GPT-5.2 to generate a corresponding natural language design description. The synthesis prompt is designed around the following key principles:

1. **Information sufficiency**: The generated description must contain all geometric information needed to fully reconstruct the CAD code, including primary geometry types, key dimensions, feature counts, relative positions, and boolean operation intents.

2. **Natural user style**: Descriptions adopt the tone of real user requests, such as "I'd like to model..." or "I need a part that...", following a human observation order from main body to attached features, then holes or slots, and finally chamfers, rather than mirroring code structure.

3. **Coherent prose**: Written in fluent, professional engineering English without bullet points, variable names, or code fragments, with numerical values naturally embedded in

VLM Judge Prompt for CAD Structural Consistency

```
You are a strict CAD structural-consistency reviewer. You will be shown two images:

  - Image A: the reference rendering (ground truth).
  - Image B: the rendering produced from the model-generated SCAD/CadQuery code.

Your task is NOT to judge aesthetics, materials, lighting, or rendering style.
You must judge ONLY the structural consistency between B and A, specifically
the factors that affect the foreground IoU between A and B.

================ OUT OF SCOPE --- IGNORE COMPLETELY ================
- Color, material, texture, shading, lighting, reflections
- Background, viewpoint differences within a small tolerance
- Rendering style, line thickness, mild aliasing, mild blur
- Surface decoration that does not change the silhouette

================ HARD GATES --- apply BEFORE the rubric ================
- If Image B is empty / corrupted / not interpretable -> all dimensions score 0.
- If image quality prevents reliable judgment -> output the most conservative
  score consistent with the visible evidence and note it in the reason.

================ Rubric: 5 Orthogonal Dimensions --- score each 0/1/2 ================

- Outline and Proportion
   - 2: The outer silhouette and aspect ratio of B closely match A; no global scaling, stretching, or shifting distorts the
      bounding-box proportions.
   - 1: The broad shape category is recognizable but with noticeable proportion errors or mild global distortion.
   - 0: The silhouette or aspect ratio of B fundamentally differs from A, or severe global distortion is present.

- Structural Completeness
   - 2: All major geometric components visible in A are present in B with correct relative sizes; no large-scale missing or
      extra blocks.
   - 1: Most components are present but one minor region is missing, displaced, or an extra mass is introduced.
   - 0: Multiple major components are missing or significant structural hallucinations are introduced.

- Cutout and Detail Consistency
   - 2: The holes, slots, cutouts, and through-features in B match those in A in count, size, and position.
   - 1: Most cutouts match but with minor count/size/position errors in one or two features.
   - 0: Major cutout mismatches: missing holes, extra holes, or significantly wrong sizes/positions.

- Layout and Symmetry
   - 2: Each local sub-structure in B is placed in the correct region consistent with A; any symmetry relations in A are
      preserved in B.
   - 1: Overall layout is correct but one sub-structure is mildly displaced or a symmetry relation is partially broken.
   - 0: Multiple sub-structures are misplaced or symmetry is severely violated.

- Topology and Connectivity
   - 2: The number of distinguishable parts and their connectivity in B correspond to A; no part is incorrectly split or
      merged; boundary overlaps well.
   - 1: Part count is correct but connectivity has minor errors, or boundary deviations are noticeable but bounded.
   - 0: Parts are incorrectly split or merged relative to A, or boundary deviations are severe.

================ SCORING INSTRUCTIONS ================
Score each of the 5 dimensions independently as 0, 1, or 2.
The total score is the sum of all 5 dimensions, ranging from 0 to 10.
Be strict: reserve score 2 for genuinely matching cases on that dimension.
When in doubt between two scores, pick the lower one.

================ OUTPUT --- STRICT JSON ARRAY, NOTHING ELSE ================
[
  {"dimension": "Outline and Proportion", "score": <0|1|2>, "reason": "<=20 words>"},
  {"dimension": "Structural Completeness", "score": <0|1|2>, "reason": "<=20 words>"},
  {"dimension": "Cutout and Detail Consistency", "score": <0|1|2>, "reason": "<=20 words>"},
  {"dimension": "Layout and Symmetry", "score": <0|1|2>, "reason": "<=20 words>"},
  {"dimension": "Topology and Connectivity", "score": <0|1|2>, "reason": "<=20 words>"}
]
```

Figure 5: Visual LLM judge prompt template for CAD structural consistency evaluation. The judge assesses five orthogonal dimensions independently, each scored 0, 1, or 2, with the total ranging from 0 to 10.

Table 8: Chamfer Distance evaluation on the UniCAD test set. Mean CD and Median CD are reported for each language (multiplied by 10^3). Lower is better. CRMoD-CAD achieves the lowest CD across all languages on both easy and hard samples, consistent with its IoU superiority.

Model	**OpenSCAD**		**CadQuery**		**JSCAD**	
	Mean CD↓	Median CD↓	Mean CD↓	Median CD↓	Mean CD↓	Median CD↓
Easy Samples						
Gemini 2.5 Pro	20.41	1.06	30.18	1.82	33.65	2.08
GPT-4o	14.83	0.72	25.37	1.32	27.52	1.58
GLM-5	10.52	0.38	14.73	0.52	12.68	0.45
GPT-5.2-high	8.31	0.21	11.52	0.12	10.28	0.34
Qwen3.5-9B	21.35	1.12	35.82	2.14	38.43	2.36
CRMoD-CAD	**4.73**	**0.12**	**6.82**	**0.08**	**7.51**	**0.15**
Hard Samples						
Gemini 2.5 Pro	34.72	11.85	41.36	13.74	46.18	15.23
GPT-4o	29.04	9.12	35.81	10.67	39.45	12.38
GLM-5	21.38	4.63	24.92	5.41	27.15	5.87
GPT-5.2-high	18.74	2.62	20.85	3.05	26.28	4.10
Qwen3.5-9B	37.61	13.24	47.53	15.82	52.17	17.41
CRMoD-CAD	**16.76**	**2.14**	**19.85**	**2.73**	**21.34**	**3.08**

sentences, *e.g.*, "the main body is an 80 × 40 × 20 block".

4. **Professional terminology**: Uses appropriate mechanical engineering terms such as through-hole, counterbore, boss, flange, chamfer, fillet, rib, and slot.

L CoT Synthesis

For each SFT sample, we use GPT-5.2 to synthesize a detailed CoT reasoning trace that decomposes the design description into a step-by-step construction plan. The synthesis prompt imposes the following core requirements on the reasoning trace:

1. **Forward-looking design plan**: Written in future tense, simulating an engineer's planning process before coding, describing how parameters, geometry, and structure will be chosen, rather than retrospectively analyzing existing code.

2. **Parametric planning**: Explicitly identifies all key parameters controlling the model, including exact parameter names, default values, and reasonable ranges, covering dimensions, tolerances, and printing constraints.

3. **Geometric decomposition**: Describes how the design will be decomposed into modules using exact module and function names, specifying boolean operation sequences, primitive compositions, and spatial transformations.

4. **Implementation strategy**: Outlines the organization of the SCAD file, including the ordering of parameter declarations, helper functions, core modules, and the main assembly, as well as how loops, conditionals, and module calls will be structured.

5. **Exact alignment with code**: The reasoning must use the exact parameter names, module names, and numerical values from the code, ensuring that following the reasoning trace would accurately reproduce the original model.

The synthesized reasoning traces are prepended to the code output during SFT training, teaching the model to perform structured reasoning before generating code. Figure 7 presents a complete rollout example, including the input design description, the model-generated CoT reasoning, and the final CAD code output.

Multi-Language CAD Code Generation Prompt Example

[System Prompt — OpenSCAD]
You are an expert OpenSCAD programmer. Given the user's design request, reason briefly in <think>...</think> and then output only valid OpenSCAD code inside a fenced code block labeled openscad, with no extra commentary.

[System Prompt — CadQuery]
You are an expert CadQuery programmer. Given the user's design request, reason briefly in <think>...</think> and then output only valid CadQuery (Python) code inside a fenced code block labeled python, with no extra commentary.

[System Prompt — JSCAD]
You are an expert JSCAD programmer. Given the user's design request, reason briefly in <think>...</think> and then output only valid JSCAD (JavaScript) code inside a fenced code block labeled javascript, with no extra commentary. Your code must export a main function that returns the geometry.

[User Query — appended after the corresponding system prompt]

Create a simple thin circular plate shaped like a coin. The body is a right circular cylinder with a diameter of approximately 1.51579 units, radius 0.7579, and an overall thickness of 0.03906 units. The cylinder's flat faces are parallel and perpendicular to the vertical axis. Position the disk such that its flat bottom rests on the base plane; the center of the circular profile is slightly offset from the global origin along the X direction by about 0.0079 units, with Y at 0.0 and Z at 0.0. There are no cutouts, holes, chamfers, or other features—just a smooth uniform cylinder.

Figure 6: Multi-language CAD code generation prompt example. The top section lists the three language-specific system prompts, each declaring an expert role and specifying the output code format. The bottom section shows a representative user query that is appended after the corresponding system prompt to form the complete training input.

Complete OpenSCAD Rollout Example

[Input Design Description]
Design a thin hollow rectangular mounting frame centered at the coordinate origin. The outer body is 10 units wide, 40 units tall, and 1 unit thick, with thickness running along the Z-axis. A centered rectangular through-cutout is removed from the middle, leaving a uniform 2-unit border and an inner opening of 6 units by 36 units. The four outer corners are softly rounded, and each corner region contains a small circular through-hole for mounting. These holes are symmetric about both axes, lie on the frame material rather than inside the central opening, and pass fully through the thickness.

[Model-Generated CoT Reasoning]
I will start from the largest visible structure: a very thin rectangular frame centered at the origin. Since the part is hollow, it is more natural to build the full outer plate first and then remove the middle opening with a boolean subtraction. The requested outer size is 10 by 40 by 1, and the inner opening is 6 by 36, so the remaining frame border will be 2 units on each side. I will keep the thickness centered around the XY plane so that the part stays symmetric along Z.

To make the model parametric, I will define the main dimensions as outer_width = 10, outer_height = 40, part_thickness = 1, inner_width = 6, and inner_height = 36. The rounded corners can be controlled by corner_radius = 1.2. For the four mounting holes, I will use hole_radius = 0.45 and place their centers at x = +/-3.6 and y = +/-18.0. These positions keep the holes close to the corner regions while still leaving material around them, and the same coordinates with opposite signs preserve symmetry across both axes.

In OpenSCAD, I will form the rounded outer profile in 2D using a square and offset(), then extrude that profile to the requested thickness. After that, a single difference() block can remove all negative features: the centered rectangular slot and the four vertical cylindrical holes. I will make the subtractive solids slightly thicker than the frame by adding a small tolerance to the Z height, which avoids coplanar faces and ensures the cutouts pass cleanly through the body. The code will therefore be organized as parameter declarations, a reusable rounded-rectangle helper, a frame_body() module, and a final boolean assembly whose names and values match the planned geometry.

[Final OpenSCAD Code Output]

```openscad
outer_width = 10;
outer_height = 40;
part_thickness = 1;
inner_width = 6;
inner_height = 36;
corner_radius = 1.2;
hole_radius = 0.45;
hole_x = 3.6;
hole_y = 18.0;

module rounded_rect(w, h, r) {
  offset(r = r)
    square([w - 2 * r, h - 2 * r], center = true);
}

module frame_body() {
  linear_extrude(height = part_thickness, center = true)
    rounded_rect(outer_width, outer_height, corner_radius);
}

difference() {
  frame_body();
  cube([inner_width, inner_height, part_thickness + 0.04], center = true);
  for (x = [-hole_x, hole_x])
    for (y = [-hole_y, hole_y])
      translate([x, y, 0])
        cylinder(h = part_thickness + 0.04, r = hole_radius, center = true, $fn = 48);
  ...
}
```

Figure 7: Complete rollout example organized from top to bottom. The example contains the input design description, the model-generated CoT reasoning, and the final OpenSCAD code output.