GEOMETRY BEFORE SYMBOLS: LATENT STATE-TRANSITION MODELING FOR AUTOREGRESSIVE CAD GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent image-to-CAD methods increasingly formulate CAD reconstruction as autoregressive generation of executable feature sequences. However, the next feature is predicted in symbolic sequence space, while its semantics are ultimately defined by the geometric state transition it induces. We introduce **ForeCAD**, a framework for latent geometric state-transition modeling in autoregressive CAD generation. Before generating each feature, ForeCAD forms internal representations of the current construction state and the anticipated feature-induced transition, so that the symbolic feature is generated as the realization of an internal geometric decision. We ground these representations with execution-derived supervision of intermediate states, material additions and removals, and state–transition consistency. We further introduce **Representation-Guided Execution Alignment (RGEA)**, which aligns generated feature consequences with the geometric intent encoded by the learned representations. To support feature-aligned supervision and evaluation, we construct **FeatureState-CAD**, spanning Procedural Construction and DeepCAD-Complex. ForeCAD consistently outperforms sequence-based and geometry-aware baselines; on DeepCAD-Complex, it improves IoU from 36.86% to 39.53%, execution rate from 88.25% to 90.00%, and mean CD from 0.090 to 0.079. Controlled ablations and representation interventions further validate the learned latent state–transition representations and the benefit of aligning their geometric intent with executable feature generation.

1 INTRODUCTION

Computer-aided design (CAD) represents 3D objects through editable construction histories composed of parameterized sketches, feature operations, and Boolean operations (Willis et al., 2021; Wu et al., 2026). Reflecting this procedural structure, recent image-to-CAD methods increasingly formulate CAD reconstruction as autoregressive generation of executable feature sequences from visual observations (Qin et al., 2025; Wang et al., 2025a). In this formulation, each generated feature is not only a symbolic continuation of the construction history, but also an executable decision that transforms the current solid into a new geometric state.

Standard autoregressive generation, however, predicts the next feature directly in symbolic sequence space. This creates a mismatch between how a CAD decision is generated and what gives that decision meaning: a feature is represented by operation tokens and parameters, while its semantics are ultimately determined by the geometric state transition it induces. The mismatch is especially pronounced under token-level supervision. A feature's geometric effect emerges from the joint execution of its operation type and parameters, yet autoregressive training decomposes this decision into individual token predictions. Consequently, small symbolic changes can produce disproportionately different geometric outcomes: replacing JOIN with CUT modifies only a small part of the command sequence, but reverses the operation from adding material to removing it. This suggests that next-feature generation should model not only which symbols come next, but also what geometric state transition those symbols are intended to realize.

Recent work has begun to incorporate geometry into CAD generation beyond sequence likelihood. One line provides geometric cues or intermediate construction states as context for subsequent gen-

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

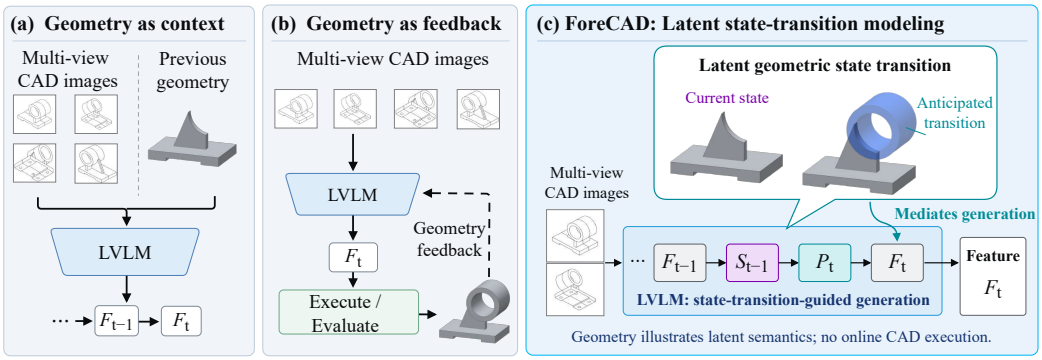


Figure 1: From external geometry to latent state-transition modeling. Existing approaches use geometry mainly as conditioning or post-generation feedback. ForeCAD instead models the current state and anticipated transition as latent representations before feature generation, making the next symbolic feature a realization of an internal geometric decision.

eration (Khan et al., 2024; Rukhovich et al., 2025; Tan et al., 2026; Feng et al., 2026), while another uses geometric outcomes to evaluate, select, or refine generated predictions (Giannone et al., 2026; Kabisov et al., 2026b). These approaches demonstrate the value of geometry, but geometry remains largely external to the feature-level autoregressive decision. As illustrated in Fig. 1, the next feature is still primarily produced as a symbolic continuation, without explicitly modeling, inside the causal generation process, the current geometric state together with the geometric change that the feature is intended to produce. The coupling between symbolic feature generation and geometric state-transition modeling therefore remains largely implicit.

Motivated by this view, we introduce **ForeCAD**, a framework for latent geometric state-transition modeling in autoregressive CAD generation. Before generating each feature, ForeCAD forms internal representations of both the current construction state and the anticipated feature-induced geometric transition. The next feature is then generated conditioned on these representations, so that feature prediction is mediated by what has already been constructed and how the geometry is intended to change. We ground the latent representations with execution-derived supervision of intermediate states, material additions and removals, and their state–transition consistency. Grounding the intended transition, however, does not by itself ensure that the generated symbolic operation realizes it. We therefore introduce **Representation-Guided Execution Alignment (RGEA)**, which uses feature-level execution feedback to align the realized consequence of a generated feature with the geometric transition encoded by the latent representations. At inference time, these representations are formed and used directly by the autoregressive model.

To provide feature-aligned supervision and evaluate this formulation across complementary CAD settings, we construct **FeatureState-CAD**, spanning Procedural Construction and DeepCAD-Complex, which emphasizes structurally complex construction programs from the DeepCAD corpus. ForeCAD consistently outperforms sequence-based and geometry-aware baselines on both settings. In particular, on DeepCAD-Complex, it improves the best competing IoU from 36.86% to 39.53%. Controlled ablations and representation interventions further show that these gains arise from latent geometric state-transition modeling and from aligning generated feature consequences with the geometric intent encoded by the learned representations.

Our main contributions are summarized as follows:

- We identify a mismatch in autoregressive CAD generation: the next feature is predicted as a symbolic continuation, while its semantics are defined by the geometric state transition it induces. This motivates a latent state-transition formulation in which the intended geometric change is modeled within the causal generation process before its symbolic realization.
- Building on this view, we develop **ForeCAD**, which models the current geometric state and anticipated transition as latent representations inside the causal generation process, grounds them with execution-derived geometry and state–transition consistency, and introduces **Representation-Guided Execution Alignment (RGEA)** to align realized feature consequences with latent geometric intent.

- We construct **FeatureState-CAD**, spanning controlled Procedural Construction and DeepCAD-Complex, and provide comprehensive evaluations showing consistent improvements over sequence-based and geometry-aware baselines. Controlled ablations and representation interventions further validate the learned latent state–transition representations and their role in guiding executable CAD generation.

2 Related Work

2.1 Image-Conditioned CAD Program Generation

Visual CAD reconstruction has progressed from specialized sketch-based systems to general executable program generation. Sketch2CAD predicts context-dependent modeling operations from user sketches and a partial CAD model (Li et al., 2020), while Free2CAD converts ordered freehand strokes into simplified CAD command sequences (Li et al., 2022). PlankAssembly reconstructs domain-specific plank programs from three orthographic drawings (Hu et al., 2023). These works demonstrate the benefit of procedural outputs, but rely on interactive inputs, stroke order, or restricted program spaces.

Recent methods increasingly generate complete CAD programs from images or engineering drawings (Xu et al., 2024). CAD2Program autoregressively produces textual shape programs from rasterized CAD drawings (Wang et al., 2025b); CAD-GPT models 3D positions, sketch planes, and planar parameters with a multimodal language model (Wang et al., 2025a); and Drawing2CAD predicts parametric operation sequences from vectorized engineering drawings (Qin et al., 2025). CADCrafter extends reconstruction to unconstrained images through geometry-conditioned diffusion (Chen et al., 2025), while Img2CAD factorizes program structure and continuous-attribute prediction (You et al., 2025). Img2CADSeq further combines hierarchical CAD sequence representations with a coarse-to-fine point-cloud intermediate for diffusion-based generation (Tan et al., 2026).

2.2 Geometry-Aware and Execution-Guided CAD Generation

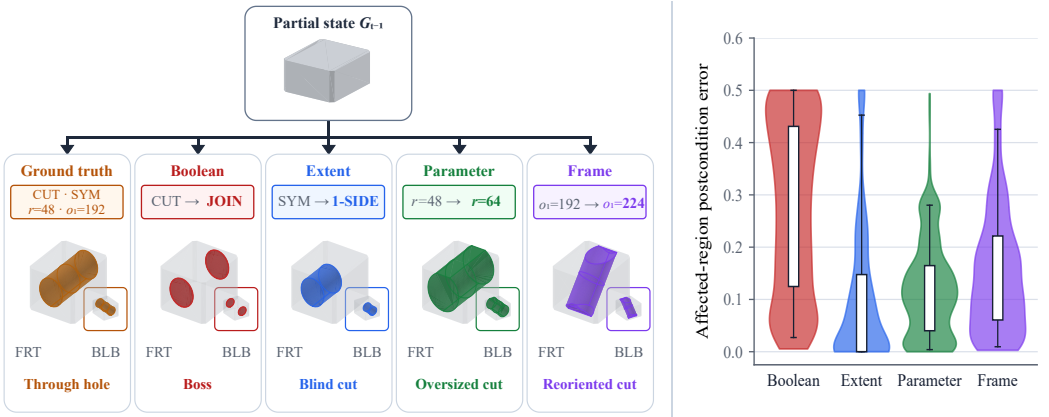
Recent CAD generation methods increasingly exploit geometry beyond symbolic sequence supervision. CADCrafter conditions generation on estimated depth and surface normals (Chen et al., 2025), while Img2CADSeq uses an intermediate point-cloud representation to bridge visual observations and CAD sequences (Tan et al., 2026). SOV-CAD conditions each modeling step on the target views, current partial model, and active sketch (Feng et al., 2026). A3PL instead introduces token-wise geometric supervision by mapping numerical CAD arguments to their induced 3D points (Guan & Sun, 2026). These methods incorporate geometry through input conditioning, intermediate geometric representations, partial construction states, or direct grounding of symbolic parameters.

A complementary direction introduces geometry through execution and feedback. GIFT executes sampled programs and uses geometric verification for candidate selection and failure-driven augmentation (Giannone et al., 2026). CADReasoner iteratively edits generated programs according to geometric discrepancies between the reconstruction and the target (Kabisov et al., 2026a). CADENA grows programs feature by feature under explicit geometric feedback and further optimizes generation with execution-based rewards (Kabisov et al., 2026b). These approaches use realized geometry to verify, refine, or optimize symbolic predictions.

3 ForeCAD: Latent Geometric State-Transition Modeling

3.1 From Symbolic Prediction to Geometric State Transitions

In autoregressive CAD generation, a feature is predicted in symbolic sequence space, while its geometric meaning is realized through the state transition it induces on the current solid. To illustrate the resulting mismatch, we perform controlled one-field perturbations on ground-truth features while keeping the same partial construction state G_{t-1}. As shown in Fig. 2, changing only the Boolean type, extrusion extent, geometric parameter, or reference frame can produce substantially different 3D outcomes and postcondition errors. Thus, even perturbations with the same symbolic edit cardinality can have markedly different geometric consequences.



(a) Controlled one-field perturbations yield distinct 3D outcomes

(b) Equal symbolic distance, unequal execution error

Figure 2: Equal symbolic edits can induce unequal geometric consequences. Starting from the same partial state, changing a single semantic field produces substantially different executed geometries and postcondition errors.

We therefore view the geometric state transition induced by a feature as its execution-level semantics. This motivates explicitly modeling the corresponding state transition within the autoregressive decision process before predicting its symbolic realization. To formalize the executed transition, let

$$G_t = \mathcal{K}(G_{t-1}, F_t), \qquad G_0 = \varnothing, \tag{1}$$

where F_t is the t-th feature and $\mathcal{K}$ denotes the CAD execution kernel.

For a query point $q \in \mathbb{R}^3$, let $O_t(q) \in \{0, 1\}$ denote the occupancy of G_t. The net material addition and removal induced by F_t are defined as

$$A_t(q) = O_t(q)\big(1 - O_{t-1}(q)\big), \qquad R_t(q) = O_{t-1}(q)\big(1 - O_t(q)\big). \tag{2}$$

They satisfy

$$O_t(q) = O_{t-1}(q) + A_t(q) - R_t(q). \tag{3}$$

3.2 State and Predictive Transition Representations

Following Eq. 3, ForeCAD models the geometric state-transition decision internally before generating its symbolic realization. We represent the inherited construction state and the anticipated feature-specific geometric change separately. We introduce a state representation S_{t-1} for the geometry constructed before F_t, and a predictive transition representation P_t for the net geometric change expected from the upcoming feature.

Both representations are inserted directly into the causal generation sequence:

$$\cdots, F_{t-1}, S_{t-1}, P_t, F_t, S_t, \cdots. \tag{4}$$

Importantly, P_t is formed before any content of F_t is generated, while S_t is formed only after the feature is completed. The current feature is therefore generated as

$$p_\theta(F_t \mid I, F_{<t}, S_{t-1}, P_t), \tag{5}$$

where I denotes the visual observations. Thus, the next symbolic feature is generated through an internal geometric state-transition decision rather than directly from the symbolic history alone.

S and P are not precomputed geometric inputs, but contextualized internal representations produced by the autoregressive backbone. By placing S_{t-1} and P_t immediately before F_t, the generation of each feature can directly condition on both the current construction state and the geometric change anticipated at that step, rather than relying on the symbolic history alone. Their geometric semantics are learned through execution-derived supervision, described next.

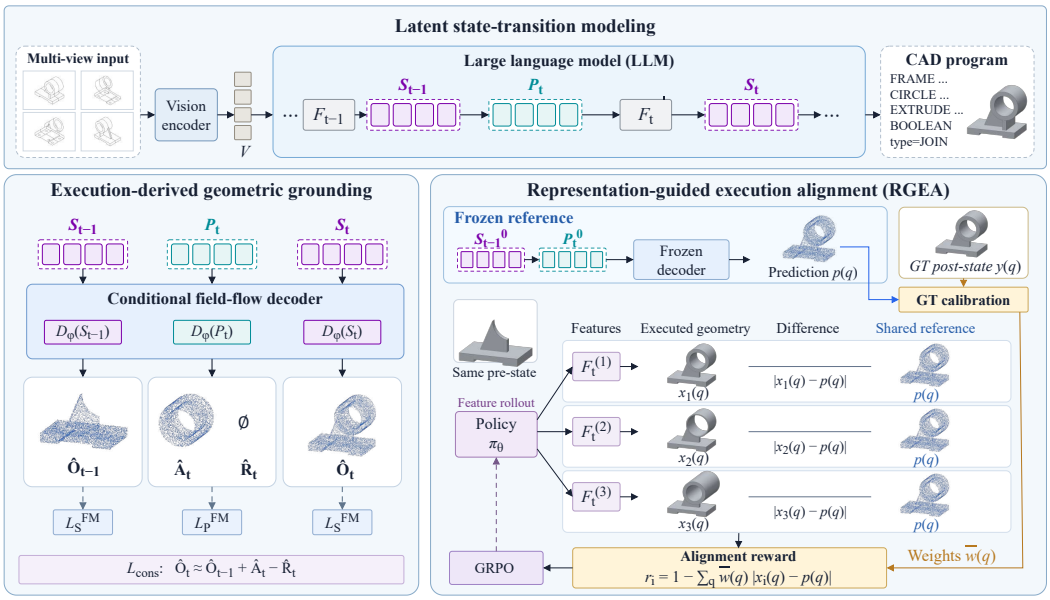


Figure 3: Overview of ForeCAD. ForeCAD instantiates latent geometric state-transition modeling within autoregressive CAD generation. It forms the state representation S_{t-1} and predictive transition representation P_t inside the causal generation process, grounds their geometric semantics with execution-derived supervision and state–transition consistency, and aligns generated feature consequences with the geometric intent through Representation-Guided Execution Alignment.

3.3 Execution-Derived Geometric Grounding

To endow the latent state-transition representations with explicit geometric semantics, we ground them using execution-derived geometry. We replay each ground-truth CAD program offline and query consecutive solids at shared spatial locations q, obtaining the pre-state $O_{t-1}(q)$, post-state $O_t(q)$, and the corresponding feature-induced addition and removal regions $A_t(q)$ and $R_t(q)$. These execution-derived fields provide direct supervision for the learned representations:

$$S_{t-1} \to O_{t-1}, \qquad P_t \to [A_t, R_t], \qquad S_t \to O_t. \tag{6}$$

We decode each representation into its corresponding geometric field and optimize the decoder with conditional geometric-field flow matching. For a target field value y, the grounding loss is defined as

$$\mathcal{L}_{\mathrm{FM}}(H, y) = \mathbb{E}\left[\|v_\phi(q, x_\tau, \tau; H) - (y - \epsilon)\|_2^2\right], \tag{7}$$

where $H \in \{S, P\}$, $x_\tau = (1-\tau)\epsilon + \tau y$, $\epsilon \sim \mathcal{N}(0, I)$, and $\tau \sim \mathcal{U}(0, 1)$. We apply this objective to the state occupancy field and the two-channel transition field, yielding $\mathcal{L}_S$ and $\mathcal{L}_P$.

Beyond individual field supervision, we enforce the geometric consistency between the predicted state and transition fields. Using the decoded clean fields $\widetilde{O}_{t-1}, \widetilde{A}_t, \widetilde{R}_t, \widetilde{O}_t$, we impose

$$\mathcal{L}_{\mathrm{cons}} = \mathbb{E}_q\left[\left(\widetilde{O}_t(q) - \widetilde{O}_{t-1}(q) - \widetilde{A}_t(q) + \widetilde{R}_t(q)\right)^2\right]. \tag{8}$$

This couples the pre-state, transition, and post-state predictions in geometric space rather than imposing a direct relation on the hidden representations.

Because feature-induced changes are often spatially localized, uniform queries can be dominated by unchanged regions. We therefore preserve global spatial coverage while allocating additional queries to transition regions and their boundaries.

The supervised objective is

$$\mathcal{L}_{\mathrm{sup}} = \mathcal{L}_{\mathrm{CAD}} + \lambda_S \mathcal{L}_S + \lambda_P \mathcal{L}_P + \lambda_{\mathrm{cons}} \mathcal{L}_{\mathrm{cons}}, \tag{9}$$

where $\mathcal{L}_{\text{CAD}}$ is the standard autoregressive CAD sequence loss. We set $\lambda_S = \lambda_P = 0.1$ and $\lambda_{\text{cons}} = 0.01$ in all experiments. The state and transition grounding objectives encourage S and P to capture the geometry of the current construction state and the feature-induced change, while the consistency constraint further requires these representations to form a coherent state transition in geometric space.

3.4 Representation-Guided Execution Alignment

Geometric grounding ensures that S and P encode the construction state and feature-induced transition, but it does not guarantee that the generated symbolic feature realizes the represented geometric intent. We therefore introduce Representation-Guided Execution Alignment (RGEA) to align feature generation with the grounded state–transition representations.

For a fixed ground-truth prefix

$$C_t = (I, F_{<t}^{GT}),$$

the current policy samples a group of candidate features

$$F_t^{(1)}, \ldots, F_t^{(K)}.$$

Each candidate is executed from the same pre-feature state G_{t-1}, producing an executed post-state $G_t^{(i)}$. Meanwhile, a frozen reference model decodes the grounded state and transition representations into a representation-derived post-operation occupancy:

$$p(q) = \text{clip}\left(\hat{O}_{t-1}(q) + \hat{A}_t(q) - \hat{R}_t(q), 0, 1\right). \tag{10}$$

Let $x_i(q)$ denote the occupancy obtained by executing candidate $F_t^{(i)}$, and $y(q)$ denote the ground-truth post-state. Instead of directly optimizing candidates toward the ground-truth geometry, RGEA uses the ground truth only to estimate the reliability of the representation-derived prediction:

$$w(q) \propto w_0(q) m(q) \left[\delta + (1-\delta)\left(1 - |p(q) - y(q)|\right)^2\right], \tag{11}$$

where $w_0(q)$ is the base spatial weight and $m(q)$ is the validity mask.

The reward for candidate i is defined by its agreement with the representation-derived geometric intent:

$$r_i = 1 - \sum_q w(q)|x_i(q) - p(q)|. \tag{12}$$

Therefore, candidates are favored when their realized execution consequence matches the grounded state–transition representation in regions where this representation is supported by the ground-truth geometry.

We optimize the policy using group-relative advantages derived from $\{r_i\}_{i=1}^K$ under the GRPO objective. RGEA updates the language adapters and state–transition embeddings while keeping the geometric decoder and reference model fixed. At inference time, candidate execution and geometric decoding are removed, and ForeCAD performs standard autoregressive generation using the learned state and transition representations.

4 FeatureState-CAD Dataset

Training ForeCAD requires feature-aligned supervision of construction states and feature-induced geometric changes. We therefore construct **FeatureState-CAD**, which augments two complementary CAD sources with execution-derived feature-level annotations. Each example pairs four fixed-view line drawings with an executable CAD program and its pre/post geometry at each feature boundary. These geometric annotations are used only during training; inference requires only the input images.

Table 1: IoU and Exec. are percentages; CD_{mean} and CD_{med} are dimensionless normalized distances reported over valid samples. Each evaluation setting contains 800 samples. Best results are shown in bold and second-best results are underlined. * denotes baselines trained on FeatureState-CAD under our unified experimental protocol.

	Procedural				DeepCAD-Complex			
Method	IoU ↑	Exec. ↑	CD_{mean} ↓	CD_{med} ↓	IoU ↑	Exec. ↑	CD_{mean} ↓	CD_{med} ↓
Sequence-Based Baselines								
Qwen3-VL-4B* (2025)	78.96	97.75	0.028	0.014	23.40	63.38	0.113	0.062
InternVL3-8B* (2025)	80.15	98.13	0.026	0.012	25.40	66.88	0.108	0.060
CAD2Program* (2025b)	75.36	94.75	0.036	0.017	20.25	57.13	0.134	0.076
CAD-GPT* (2025a)	79.42	96.13	0.030	0.014	24.82	64.25	0.119	0.067
PlankAssembly* (2023)	72.81	92.75	0.041	0.019	16.75	48.88	0.152	0.087
Geometry-Aware Baselines								
A3PL* (2026)	83.68	98.38	0.024	<u>0.011</u>	32.35	77.13	0.101	0.055
SOV-CAD* (2026)	<u>85.73</u>	99.00	<u>0.021</u>	**0.010**	36.41	87.13	<u>0.090</u>	<u>0.048</u>
GIFT* (2026)	85.29	<u>99.13</u>	0.022	**0.010**	<u>36.86</u>	<u>88.25</u>	0.092	0.050
Ours								
ForeCAD w/o RGEA	84.77	98.50	<u>0.021</u>	<u>0.011</u>	32.55	77.25	0.099	0.049
ForeCAD	**87.30**	**99.88**	**0.019**	<u>0.011</u>	**39.53**	**90.00**	**0.079**	**0.042**

Data sources. FeatureState-CAD contains two complementary settings: Procedural Construction and DeepCAD-Complex. Procedural Construction contains 24 rule-generated mechanical-part families with structured construction rules and constraint-preserving parametric variation. DeepCAD-Complex is a complexity-focused setting derived from the original DeepCAD partitions (Wu et al., 2021). Because DeepCAD contains many elementary construction histories, evaluating on the full distribution can cause simple cases to dominate aggregate metrics and reduce the sensitivity of the benchmark to differences among methods. We therefore emphasize structurally complex programs, including richer feature sequences, sketches, Boolean operations, reference frames, and multi-loop or non-one-sided constructions, so that the evaluation more clearly distinguishes model performance on demanding CAD construction histories. Both settings use the same CAD representation and rendering pipeline. The training set contains 58,530 programs and 176,149 features.

Execution-aligned annotations. We replay each program feature by feature to obtain pre- and post-execution solids and derive the corresponding material addition and removal regions. For each training feature, we cache 2,048 shared spatial queries with occupancy labels for the state and transition fields, with additional sampling near changed regions and boundaries.

Evaluation splits. We evaluate on a fixed set of 800 examples from each setting, with no sample-ID overlap with training. Details are provided in the supplementary material.

5 Experiments

5.1 Experimental Setup

We evaluate ForeCAD on the two FeatureState-CAD settings described in Sec. 4. We report execution rate (Exec.), volumetric intersection-over-union (IoU), and Chamfer Distance (CD_{mean} and CD_{med}), measuring program validity and geometric fidelity. All generated programs are evaluated under the same strict parsing and execution protocol; IoU is evaluated over the complete test population, while CD is reported over valid reconstructions.

We compare ForeCAD against representative sequence-based and geometry-aware CAD generation methods. ForeCAD is built on Qwen3-VL-4B-Instruct (Bai et al., 2025). All baselines are trained and evaluated on FeatureState-CAD under a unified experimental protocol with matched training data, inputs, CAD representation, and training amounts.

Table 2: Ablation of state–transition grounding. Slots denotes the insertion of S/P representations; State and Trans. denote execution-derived geometric supervision for the state and predictive transition representations, respectively; Cons. denotes the state–transition consistency objective.

				Procedural			DeepCAD-Complex		
Slots	State	Trans.	Cons.	IoU ↑	Exec. ↑	CD_{mean} ↓	IoU ↑	Exec. ↑	CD_{mean} ↓
–	–	–	–	78.77	97.63	0.027	23.10	63.13	0.116
•	–	–	–	78.59	97.50	0.028	22.65	63.38	0.120
•	•	–	–	81.38	98.38	0.021	27.51	74.63	0.104
•	–	•	–	83.76	**98.50**	**0.020**	29.61	74.63	**0.093**
•	•	•	–	84.20	98.38	0.022	31.60	75.88	0.101
•	•	•	•	**84.77**	**98.50**	0.021	**32.55**	**77.25**	0.099

Table 3: Ablation of representation-guided execution alignment. Target is GT geometry (y) or the representation prediction (p); Calib. enables reliability weighting. "–" denotes no post-training, and Rep./Yes is full RGEA.

Target	Calib.	IoU ↑	Exec. ↑	CD_{mean} ↓
Procedural				
–	–	84.77	98.50	0.021
GT	No	85.10	99.63	0.021
GT	Yes	85.78	99.63	0.020
Rep.	No	86.42	99.38	0.020
Rep.	Yes	**87.30**	**99.88**	**0.019**
DeepCAD-Complex				
–	–	32.55	77.25	0.099
GT	No	36.05	82.75	0.086
GT	Yes	37.18	85.25	0.083
Rep.	No	38.12	87.38	0.081
Rep.	Yes	**39.53**	**90.00**	**0.079**

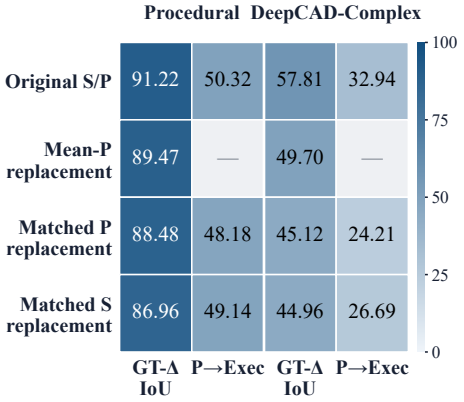


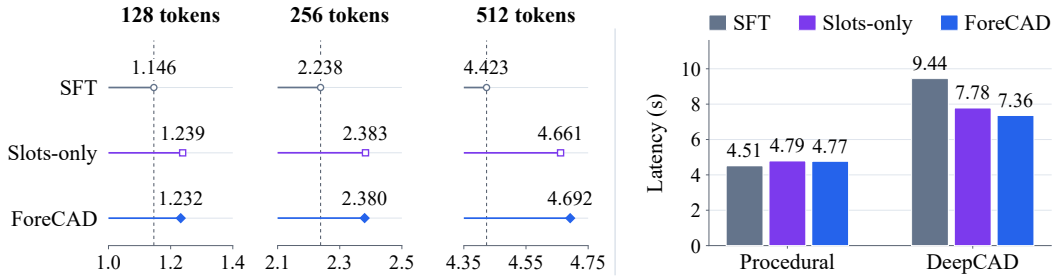
Figure 4: Representation interventions on learned latent state–transition representations. GT-ΔIoU measures agreement between the executed and ground-truth feature transitions, while P→Exec measures agreement between the transition encoded by P and the executed transition.

5.2 Main Results

Table 1 compares ForeCAD with sequence-based and geometry-aware baselines. ForeCAD achieves the best IoU, execution rate, and mean CD on both settings. On Procedural, it reaches 87.30% IoU and 99.88% execution rate, where executability is already near saturation for several baselines. The margin widens on DeepCAD-Complex: ForeCAD improves the best baseline IoU from 36.86% to 39.53% and execution rate from 88.25% to 90.00%, while reducing the best mean/median CD from 0.090/0.048 to 0.079/0.042. These results show that ForeCAD maintains a clear advantage on structurally complex CAD construction histories.

5.3 Validating State–Transition Grounding

Table 2 validates the role of state–transition grounding in ForeCAD. Adding S/P representations alone does not improve reconstruction quality, indicating that additional representation capacity is insufficient without geometric supervision. Grounding either the state or predictive transition representation substantially improves reconstruction, with transition grounding providing the stronger individual gain and lower CD. Combining state and transition supervision further improves IoU on both settings. Adding state–transition consistency further improves IoU and execution rate over the jointly supervised model while also reducing its CD. These results show that the gains come from assigning the latent state–transition representations explicit execution-derived geometric semantics and coupling them coherently, rather than from simply introducing additional representations.



(a) Matched-length latency **(b) Full-program latency**

Figure 5: **Inference efficiency of ForeCAD.** Despite modeling latent geometric state transitions, ForeCAD requires neither geometric decoding nor CAD execution during standard inference. (a) Matched-length latency for 128, 256, and 512 generated tokens. (b) Mean full-program latency under free generation on Procedural and DeepCAD-Complex.

5.4 Representation Usage and Execution Alignment

Table 3 disentangles the effects of the reward target and reliability calibration under the same supervised initialization and optimization protocol. GT-only GRPO already improves reconstruction, while either adding calibration or replacing the ground-truth target with representation-derived geometric intent yields further gains. On DeepCAD-Complex, IoU increases from 32.55% to 36.05% with GT-only GRPO, to 37.18% with calibration, and finally to 39.53% with full RGEA. These results support the contributions of both representation-derived targets and reliability calibration, with their combination achieving the best reconstruction performance among the evaluated variants.

Figure 4 evaluates whether the learned S/P representations carry instance-specific geometric information through controlled representation interventions. P→Exec is a strict hard-IoU diagnostic over signed transition fields, rather than the soft alignment reward used by RGEA, and its absolute magnitude is therefore not directly comparable to volumetric reconstruction IoU. The original S/P representations achieve the highest agreement on both settings, while mean or matched cross-example replacements consistently reduce the scores. The degradation is especially pronounced on DeepCAD-Complex: replacing P with a matched cross-example representation reduces P→Exec from 32.94 to 24.21 and GT-ΔIoU from 57.81 to 45.12. These interventions indicate that subsequent feature generation uses instance-specific geometric information carried by the learned state–transition representations.

5.5 Inference Efficiency

Figure 5 evaluates the inference overhead introduced by the internal S/P representations. At matched output lengths, ForeCAD incurs only a modest 6–8% latency increase relative to SFT. Under free generation, however, ForeCAD achieves lower full-program latency on DeepCAD-Complex despite this per-token overhead. This is because its generated programs are shorter and terminate earlier on average, offsetting the additional computation introduced by the S/P representations. Overall, latent geometric state-transition modeling improves reconstruction quality with limited per-token overhead and can reduce end-to-end generation time on complex CAD programs through more efficient program generation.

6 Conclusion

We introduced **ForeCAD**, a latent geometric state-transition framework for autoregressive CAD generation. ForeCAD models the current construction state and anticipated geometric transition before feature generation, grounds them with execution-derived geometry and state–transition consistency, and aligns symbolic outputs with the represented geometric intent through RGEA. Experiments on Procedural Construction and DeepCAD-Complex show consistent improvements over sequence-based and geometry-aware baselines, while controlled ablations and representation interventions further validate the learned state–transition representations and their role in guiding executable CAD generation.

AI USE STATEMENT

In this work, we used generative AI tools to assist with brainstorming, language editing, literature organization, and code development. All AI-assisted content, including technical ideas, text, and code, was reviewed and verified by the authors. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or sensitive user data. We are not aware of any significant ethical concerns arising from the methods, datasets, or experiments presented in this work.

REPRODUCIBILITY STATEMENT

We provide the implementation details, experimental settings, evaluation protocols, and relevant hyperparameters needed to reproduce our results in the main paper and appendix. Additional implementation details and supporting materials are provided in the supplementary material.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

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

Zhaopeng Feng, Chen Zhi, Xuhong Zhang, Zhengwen Feng, and Xinkui Zhao. Sov-cad: Stepwise orthographic views guided cad modeling sequence reconstruction. *arXiv preprint arXiv:2607.04119*, 2026.

Giorgio Giannone, Anna Clare Doris, Amin Heyrani Nobari, Kai Xu, Akash Srivastava, and Faez Ahmed. Gift: Bootstrapping image-to-cad program synthesis via geometric feedback. *arXiv preprint arXiv:2603.27448*, 2026.

Yijia Guan and Jianhua Sun. Bridging tokens and geometry: Token-wise 3d supervision for CAD generation. In *Forty-third International Conference on Machine Learning*, 2026. URL `https://openreview.net/forum?id=EwwyJYkVJV`.

Wentao Hu, Jia Zheng, Zixin Zhang, Xiaojun Yuan, Jian Yin, and Zihan Zhou. Plankassembly: Robust 3d reconstruction from three orthographic views with learnt shape programs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 18495–18505, October 2023.

Soslan Kabisov, Vsevolod Kirichuk, Andrey Volkov, Marina Barannikov, Gennadiy Savrasov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. Cadreasoner: Iterative program editing for cad reverse engineering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6143–6153, 2026a.

Soslan Kabisov, Gennadiy Savrasov, Maksim Elistratov, Antonio Rodriguez, Daniil Ignatiev, Nikita Gavrilov, Rustam Uzdenov, Alexey I Boyko, Igor Pasechnik, Anton Konushin, et al. Cadena: Stepwise cad reverse engineering. *arXiv preprint arXiv:2608.00799*, 2026b.

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

Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J. Mitra. Sketch2cad: sequential cad modeling by sketching in context. *ACM Trans. Graph.*, 39(6), November 2020. ISSN 0730-0301. doi: 10.1145/3414685.3417807. URL https://doi.org/10.1145/3414685.3417807.

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

Feiwei Qin, Shichao Lu, Junhao Hou, Changmiao Wang, Meie Fang, and Ligang Liu. Drawing2cad: Sequence-to-sequence learning for cad generation from vector drawings. In *Proceedings of the 33rd ACM international conference on multimedia*, pp. 10573–10582, 2025.

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

Shiyu Tan, Zixuan Zhao, Hao Gao, Zhiheng Chen, Xiaolong Yin, and Enya Shen. Img2cadseq: Image-to-cad generation via sequence-based diffusion. *arXiv preprint arXiv:2605.13293*, 2026.

Siyu Wang, Cailian Chen, Xinyi Le, Qimin Xu, Lei Xu, Yanzhou Zhang, and Jie Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. In *Proceedings of the AAAI conference on artificial intelligence*, volume 39, pp. 7880–7888, 2025a.

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

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

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

Yuchuan Wu, Ke Niu, Haiyang Yu, Zhuofan Chen, Xiangyang Xue, and Bin Li. Itercad: Iterative program repair for cad code generation from orthographic views. *arXiv preprint arXiv:2608.24020*, 2026.

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

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

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

A Implementation Details

This section specifies the sequence interface, trainable modules, and training schedule used to instantiate ForeCAD. We distinguish the program-level supervised stages from the feature-level alignment stage; the geometric objectives and alignment algorithm are described separately in their respective appendix sections.

A.1 CAD Representation and Input Sequence

Inputs and prediction targets. Each example contains four ordered tetrahedral line drawings and an executable CAD program. The view order is fixed to front–right–top, back–right–bottom, front–left–bottom, and back–left–top. The images and a reconstruction instruction form the user message under the native Qwen3-VL chat template; the assistant target is the CAD program. We use the following instruction:

> Reconstruct the CAD model shown in the four ordered tetrahedral views. Return only the executable TraceCAD DSL program.

Each feature is a sketch–extrusion macro-operation containing a reference frame, one or more sketch loops, an extrusion specification, and a Boolean operation. Sketch primitives include lines, arcs, and circles; Boolean operations include new body, join, cut, and intersect. Numerical geometry parameters use the dataset's 256-bin quantization and are serialized as integer fields. These integers are ordinary text, not additional learned quantization tokens. Full field definitions and normalization conventions belong to the dataset specification.

Causal insertion of representations. We preserve native tokenization of the DSL, including its potentially multi-token structural markers. Only eight internal slot identifiers are registered: four for state and four for predictive transition representations. For a program with T features, the expanded assistant sequence is schematically organized as follows:

```
Quantization header
<PROGRAM_BEGIN> [S0:  4 slots]
<FEATURE_BEGIN> [P1:  4 slots] feature 1 <FEATURE_END>
  [S1:  4 slots]
<FEATURE_BEGIN> [P2:  4 slots] feature 2 <FEATURE_END>
  [S2:  4 slots]
...
<PROGRAM_END>
```

The quantization header is part of the target program and precedes the first state block. A deterministic boundary controller inserts S_0 after the program-begin marker, P_t after the feature-begin marker, and S_t after the corresponding feature-end marker. It inserts $8T + 4$ additional sequence positions in total. Slots are processed by the causal backbone but are not sampled as output actions or written into the executable DSL.

All preceding program text remains visible under ordinary causal attention. Consequently, P_t can observe the feature-begin marker, but not the operation fields or parameters of F_t; S_t is contextualized only after F_t. Boundary detection tolerates trailing whitespace within a native token, but rejects a token that crosses into feature content before slot insertion. Neither ground-truth solids nor query-point annotations are supplied as additional visual or textual inputs.

Token supervision. Supervised training uses teacher forcing over the complete target DSL, including its header and structural markers. The prompt, image positions, padding, chat-template positions outside the DSL, and inserted slots have ignored CE labels. The standard one-position causal shift is retained: when a slot block precedes a DSL token, the final slot's hidden state predicts that token. We average CE over all non-ignored target tokens in each microbatch, rather than sampling a single feature for text supervision.

A.2 Model Architecture and Trainable Components

Backbone and adapters. ForeCAD uses Qwen3-VL-4B-Instruct with its native multimodal processing path, including visual DeepStack features and multimodal rotary positions. We freeze the original vision and language parameters, vocabulary embeddings, and language-model output head. LoRA adapters are attached only to the language model's attention projections (q, k, v, o) and feed-forward projections (gate, up, and down); no visual LoRA is used.

The four state embeddings and four transition embeddings are shared across all feature positions and examples, with separate parameters for the two groups. They are initialized from $\mathcal{N}(0, 0.02^2)$ and substituted through an input-side embedding wrapper without training the original vocabulary rows. The contextualized slot hidden states, rather than the initial embeddings alone, serve as S and P. Internal slot identifiers are masked from the language-model output distribution.

Conditional geometric-field decoder. A shared decoder projects the backbone hidden states to width 256 using layer normalization followed by a linear layer. Each point query combines its 3D coordinate, flow time, and noisy field value with a learned task embedding. Coordinates use six sinusoidal frequency bands and time uses eight, with frequencies $\pi 2^k$; raw coordinates and time are also retained. Four residual blocks apply four-head point-to-slot cross-attention and a SiLU feed-forward network of inner width 1024. Query points do not attend to one another. State and transition tasks share the slot projection and decoder blocks, but use task-specific noisy-field input and velocity-output projections. The state head outputs one occupancy velocity channel; the transition head outputs two channels for material addition and removal.

The decoder is an auxiliary training and analysis module, not an online CAD executor. Standard full-program inference retains the LoRA-adapted backbone and boundary-controlled slots, without geometric-field decoding, candidate execution, or execution-based reranking.

A.3 Training Schedule and Parameter Freezing

The implementation comprises three supervised stages followed by RGEA. Stages A and C each traverse the 58,530-program training set once; Stage B uses a fixed 500-update warm-up. Stage D instead operates on 3,000 selected training features, equally divided between the two data sources. Table 4 summarizes the schedule. Original backbone parameters remain frozen throughout.

Table 4: Training stages of the main model. $\mathcal{L}_{\text{sup}}$ is the CE-plus-geometric objective in Eq. 9. The Stage D budget counts feature groups, not complete CAD programs.

Stage	Trainable modules	Objective	Training budget
A: CAD SFT	Language LoRA	$\mathcal{L}_{\text{CAD}}$	1 epoch; 1,830 updates
B: geometry warm-up	Slot embeddings and decoder	$\mathcal{L}_{\text{sup}}$	500 updates
C: joint training	Language LoRA, slots, and decoder	$\mathcal{L}_{\text{sup}}$	1 epoch; 1,830 updates
D: RGEA	Language LoRA and slots	$\mathcal{L}_{\text{GRPO}}$	1 pass over 3,000 features; at most 188 updates

Supervised stages. Stage A does not insert slots and trains a sequence-only CAD generator. Stage B initializes the language adapters from the completed Stage A model, introduces the slot blocks, and trains the slots and geometric decoder while freezing the adapters. Importantly, Stage B retains CAD CE as well as the geometric losses: the frozen backbone remains differentiable with respect to the slot embeddings. It is therefore not a geometry-only optimization. Frozen adapters are evaluated without LoRA dropout during this stage. Stage C initializes from the completed Stage B model and jointly updates the adapters, slots, and decoder. Each stage starts with a new optimizer and learning-rate schedule; optimizer state is restored only when resuming within the same stage.

Feature-level alignment. Stage D initializes both the current policy and a separate frozen reference from the Stage C endpoint. The reference model and geometric decoder remain fixed, whereas the policy's own contextualized slots are recomputed through the differentiable prefix. Policy gra-

dients can therefore update the slot embeddings and language adapters. We sample four candidate features per prefix and process 16 prefix groups per batch. The sole objective is GRPO plus KL regularization; CE replay, FM losses, and the consistency loss are not added in Stage D. All-failure or low-reward-variance groups are excluded from the policy update, and an entirely inactive batch produces no optimizer step.

A.4 Optimization and Hyperparameters

Table 5 lists the shared supervised-stage settings. Learning rates apply only to modules enabled in the corresponding stage. We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. Each stage uses a linear learning-rate warm-up for $\lfloor 0.05N \rfloor$ updates followed by cosine decay over its own N-update horizon: 91 warm-up updates for A/C and 25 for B.

Table 5: Optimization and memory settings for Stages A–C. The sequence limit includes multimodal positions and inserted slots.

Setting	Value
LoRA rank / scaling / dropout	32 / 64 / 0.05
Peak learning rate: LoRA	10^{-4}
Peak learning rate: slots and decoder	3×10^{-4}
Weight decay / gradient-norm clipping	0.01 / 1.0
Per-GPU microbatch / gradient accumulation	8 programs / 1 step
Number of data-parallel GPUs / global batch	4 / 32 programs
Maximum sequence length	8,192 positions
Image processor minimum / maximum pixel budget	256^2 / 512^2 per image
State / transition / consistency loss weights	0.1 / 0.1 / 0.01
Cached spatial queries	2,048 per feature
Geometry query chunk / CE chunk	512 points / 256 target tokens

B FeatureState-CAD Dataset

B.1 Dataset Overview and Splits

This section expands the dataset description in Sec. 4. FeatureState-CAD pairs four fixed-view line drawings with an executable CAD program and its feature-level pre/post geometry. We use the same two settings as the main text: *Procedural Construction* and *DeepCAD-Complex*. Procedural Construction contains 24 rule-generated mechanical-part families; DeepCAD-Complex is derived from DeepCAD (Wu et al., 2021) through program-complexity filtering and execution validation. Both settings share the CAD representation and rendering pipeline. The input drawings in both settings are rendered, not photographed or scanned.

Dataset size and units. A program is one reconstruction example with four input images. A feature is one sketch–extrusion macro-operation within that program, not an individual DSL token or sketch primitive. The training set contains 58,530 programs and 176,149 features, and evaluation uses 800 examples from each setting, with no sample-ID overlap with training. Table 6 gives the per-setting counts. All supervised stages use the same training program list. Execution-derived annotations provide training supervision; full-program inference requires only the input images.

Table 6: FeatureState-CAD statistics corresponding to the main text. Counts refer to the retained training set and the evaluated 800-example subsets, not to the larger source pools used during dataset construction. Evaluation uses the fixed subsets described in Sec. 4.

Quantity	Procedural Construction	DeepCAD-Complex	Total
Training programs	29,956	28,574	58,530
Training features	90,380	85,769	176,149
Evaluation programs	800	800	1,600

Stored artifacts. Each source example includes the four images, a text DSL program, a feature-to-character-boundary map, and metadata for its origin, split, normalization, and replay. Accepted training examples additionally reference the geometric cache and audit report. The cache stores query coordinates, before/after occupancy, added/removed material labels, validity masks, and normalized loss weights. These geometric annotations are supervision, not additional model inputs at full-program inference.

B.2 Procedural Construction Dataset

Mechanical families and variation. The procedural source uses 24 mechanical-part grammars covering brackets, flanges, supports, housings, couplings, clamps, links, and related parts. Each grammar specifies a sequence of sketch profiles, reference frames, extrusion depths, and additive or subtractive operations. Variants change dimensions coherently across a family, including axis scales, profile scales, and discrete rotations, rather than perturbing every coordinate independently. Valid prefixes of longer recipes provide different feature counts. A family name therefore identifies a construction grammar; a short prefix need not contain every detail of its fully developed part. Figures 7 and 8 show one existing training example from each family.

Construction and distribution alignment. The generator uses split-level DeepCAD targets for feature count, complexity tier, Boolean schedule, and approximate DSL character length to choose compatible recipe prefixes. It does not copy the corresponding DeepCAD shape or sketch coordinates. Alignment is approximate: in particular, the procedural programs use new-body, join, and cut operations, whereas the shared representation also supports intersection. The source training programs contain 1–10 features, with a mean of 3.02 before cache filtering. Family frequencies are not uniform, since compatible families depend on the requested construction length and operation schedule.

B.3 DeepCAD-Complex Dataset

Source and quality gate. DeepCAD-Complex is derived from DeepCAD's original partitions. We require an available official quantized sequence, at least one effective sketch–extrusion feature, a non-flat 3D bounding box, and an additive first feature. Entity, profile, loop, and frame references must be complete, and sketch primitives must be supported and non-degenerate. Empty-profile extrusions are omitted consistently with the source preprocessing. The supported primitive vocabulary is line, arc, and circle.

Program-complexity gate. We apply a structural gate before expensive geometric replay. Table 7 gives the regular and strong challenge signals. A quality-valid single-feature program requires at least one strong signal. A program with two or more features requires at least two regular signals or one strong signal. This excludes elementary one-step constructions without equating complexity solely with sequence length: a dense single sketch can still qualify.

Table 7: Complexity signals for candidate DeepCAD programs. Thresholds in each column are counted as separate signals; a dash means that the property has no additional strong threshold.

Property	Regular signal	Strong signal
Effective features	≥ 3	≥ 5
Sketch primitives	≥ 10	≥ 18
Cut/intersection features	≥ 1	≥ 2
Distinct sketch frames	≥ 2	≥ 3
Hole loops	≥ 1	≥ 2
Arc/circle primitives	≥ 2	≥ 5
Multi-loop feature	Present	–
Symmetric/two-sided extrusion	Present	–
Intersection operation	–	Present

A separate weighted log score ranks the retained candidates by these structural properties. It is not the binary keep rule. The bottom 70%, next 20%, and top 10% of selected training candidates define the *medium*, *hard*, and *very hard* tiers; validation and test use the same training-derived

thresholds. Later replay and duplicate filtering can change these proportions. The tiers denote program complexity rather than measured model difficulty. Specifically, with $\ell(z) = \log(1+z)$, the score is

$$\begin{aligned}\kappa = {} & 2.5\ell(n_f) + 0.8\ell(n_p) + 1.5\ell(n_b) + 1.3\ell(\max(n_s - 1, 0)) \\ & + 1.0\ell(n_h) + 0.7\ell(n_c) + 0.7\ell(n_m) + 0.5\ell(n_e),\end{aligned} \tag{13}$$

where the counts refer to features, primitives, non-additive features, distinct frames, hole loops, curved primitives, multi-loop features, and non-one-sided extrusions, respectively. Scores are rounded to six decimals.

Replay, rendering, and deduplication. We replay the official quantized sequence and retain valid, nonempty, single-solid construction states. The resulting exact program supplies both the image geometry and the feature-boundary annotations. Figures 9 and 10 show 24 training examples from the previously prepared complexity-diverse gallery. They illustrate input variety rather than the frequency of each tier in the dataset.

B.4 Unified CAD Representation and Rendering

Executable feature interface. Both sources use the same text DSL. A feature contains a sketch frame, one or more closed loops of lines, arcs, or circles, an extrusion specification, and a Boolean operation. Numeric arguments are serialized from the 256-bin quantized representation; the DSL can be parsed back to the encoded sequence without changing it. In the replay convention, new-body and join are additive, cut removes material, and intersect retains the overlap with the extrusion tool. Only programs meeting the single-solid replay checks enter the source collection. The shared text representation avoids assigning different output languages to the two data sources.

Coordinate alignment. For rendering and spatial queries, an executed point $\mathbf{x}$ is mapped to canonical coordinates by

$$\mathbf{q} = \frac{\mathbf{x} - \mathbf{c}}{e}, \tag{14}$$

where $\mathbf{c}$ and e are the center and cube extent recorded for the final replayed shape. All intermediate states of a program use this same transform; a small hole or early construction state is not separately rescaled. This spatial transform is distinct from the field-specific parameter quantization in the DSL. In particular, the pipeline renders the quantized program's execution, so quantization does not introduce a second, inconsistent visual target.

Four fixed views. Each program provides four 384×384 orthographic visible-line drawings. The ordered viewing directions are

$$\frac{(1,-1,1)}{\sqrt{3}}, \quad \frac{(1,1,-1)}{\sqrt{3}}, \quad \frac{(-1,-1,-1)}{\sqrt{3}}, \quad \frac{(-1,1,1)}{\sqrt{3}}, \tag{15}$$

corresponding to front–right–top (FRT), back–right–bottom (BRB), front–left–bottom (FLB), and back–left–top (BLT). Their pairwise dot products are $-1/3$. The projection of the positive global vertical axis sets screen-up. Hidden lines and dimension annotations are omitted. The input images are rendered from the final shape, while supervision is obtained from the full construction history.

B.5 Execution-Derived Annotation Pipeline

Intermediate solids and net change. Given features $F_1, \ldots, F_T$, exact CAD replay produces $G_0 = \varnothing, G_1, \ldots, G_T$, where G_t is the solid after F_t. For each feature, define the net material regions

$$\mathcal{A}_t = G_t \setminus G_{t-1}, \qquad \mathcal{R}_t = G_{t-1} \setminus G_t. \tag{16}$$

These regions are geometric consequences of the operation, not the complete extrusion tool: overlap with existing material is not an addition, and empty space traversed by a cutting tool is not a removal. Figure 6 visualizes three actual annotated steps, including the empty initial state, addition, and subtraction.



Figure 6: Execution-derived ground-truth annotations. Columns show the pre-feature solid, added material, removed material, and post-feature solid. The first two rows are consecutive steps of a procedural training program; the last is an intersection that removes material from a DeepCAD-Complex training program sourced from DeepCAD. Empty regions are marked explicitly. Each program uses a fixed camera and scale across its states. Meshes visualize the CAD-derived regions; they are not model predictions or the finite spatial queries used for training.

Point labels and validity. At each sampled coordinate $\mathbf{q}$, we query before/after occupancy $O_{t-1}(\mathbf{q})$ and $O_t(\mathbf{q})$ and derive binary labels

$$A_t(\mathbf{q}) = (1 - O_{t-1}(\mathbf{q}))O_t(\mathbf{q}),$$
$$R_t(\mathbf{q}) = O_{t-1}(\mathbf{q})(1 - O_t(\mathbf{q})). \tag{17}$$

Labels are classified against CAD solids rather than inferred from a rendered silhouette. Points that cannot be classified unambiguously as inside or outside are masked out. The world-space classification tolerance is $\max(10^{-7}e, 10^{-9})$. On valid points, the targets obey $O_t = O_{t-1} + A_t - R_t$ and $A_t R_t = 0$. The two state fields and the two transition channels use the same query coordinates at each feature, making these identities pointwise, rather than comparisons between independently sampled clouds.

B.6 Spatial Query Sampling and Weighting

Global coverage and localized transitions. Uniform volume sampling alone can miss a narrow cut, thin rib, or small hole even when that change is important for the CAD program. We therefore cache one set of 2,048 distinct query coordinates per feature, combining global coverage with change-aware samples (Table 8). Coordinates can differ across features and examples; they are not point indices with a fixed semantic correspondence. All fields associated with one feature nevertheless share the same coordinates.

Table 8: Spatial-query allocation for a feature with nonzero change. "Inside" and "outside" refer to the changed region, not necessarily to the post-feature solid.

Stratum	Points	Purpose
Global	1,024	Coverage of the full spatial query domain
Change interior	256	Positive supervision within added/removed material
Boundary inside	256	Changed-side samples near the change boundary
Boundary outside	256	Unchanged-side samples across that boundary
Near unchanged	256	Local negatives around the changed region
Total	2,048	1,024 global and 1,024 transition-directed queries

Query domain and component coverage. Global points are sampled uniformly in $[-0.7, 0.7]^3$ under the original rendering normalization. If an intermediate solid exceeds this domain, the domain is expanded once for the entire program, with an 8% margin and its half-extent rounded upward in increments of 0.05. Images, DSL parameters, and the normalization transform remain unchanged. Directed quotas are divided as evenly as possible among nonempty connected components of the change region, preventing a large component from receiving all of the directed supervision.

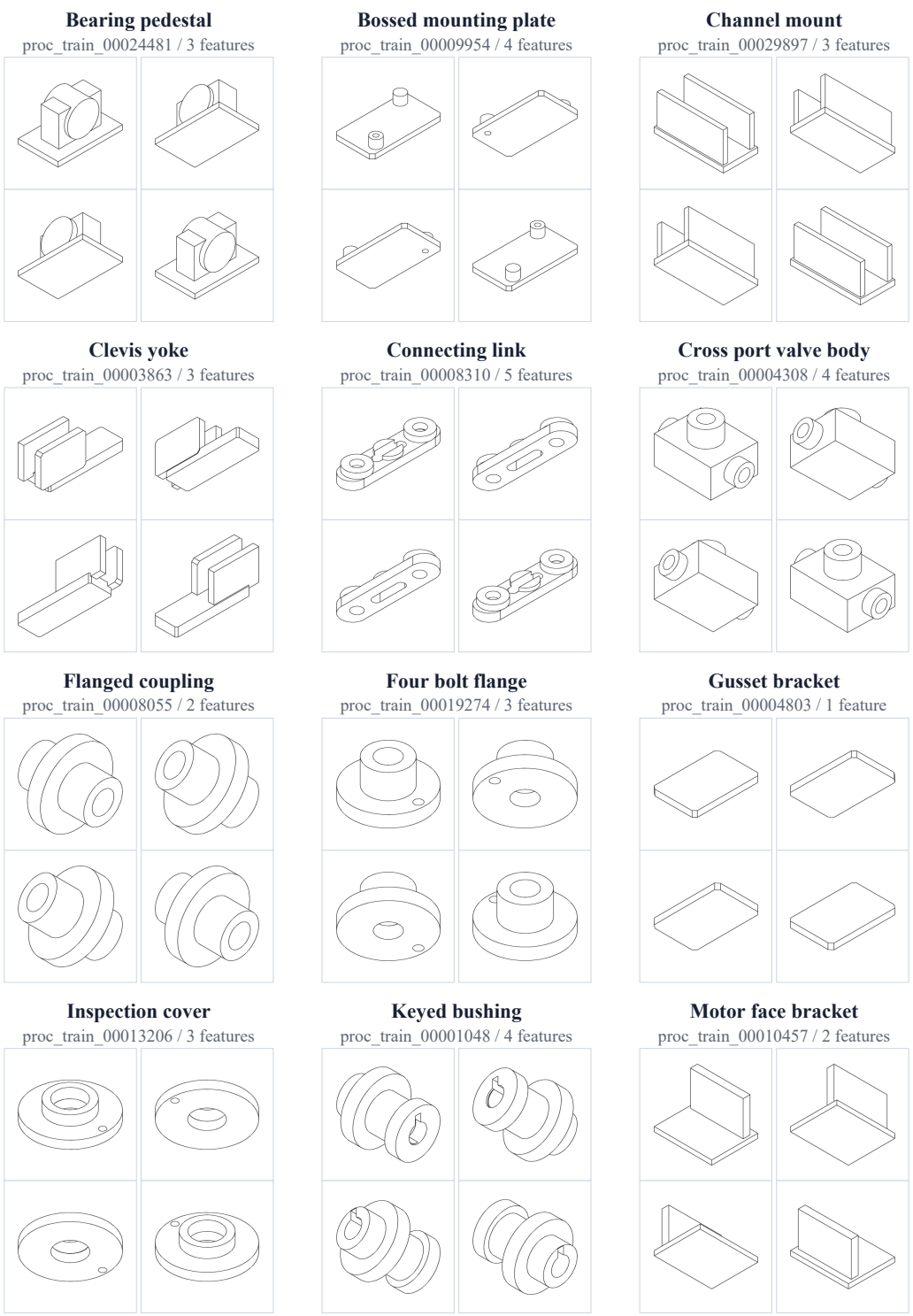


Figure 7: Procedural Construction training examples (1/2). Each card contains four original input views in row-major order: FRT, BRB, FLB, BLT. Labels identify the mechanical family, sample ID, and feature count. The two pages together cover all 24 families; these are ground-truth inputs, not generated reconstructions.

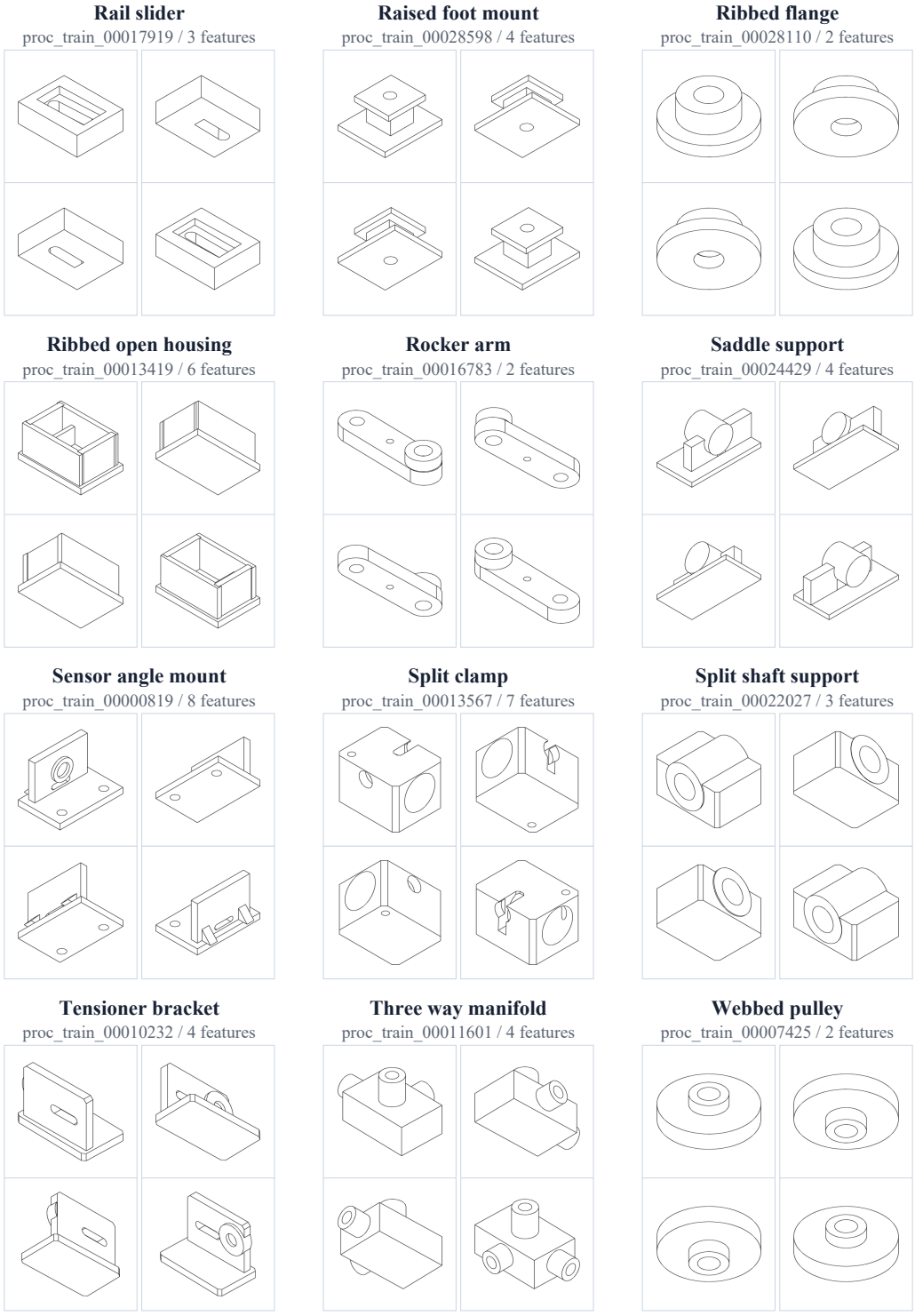


Figure 8: Procedural Construction training examples (2/2), with the same four-view arrangement as Fig. 7. The family denotes the source grammar; shorter programs can be valid construction prefixes that omit later details of the full recipe.

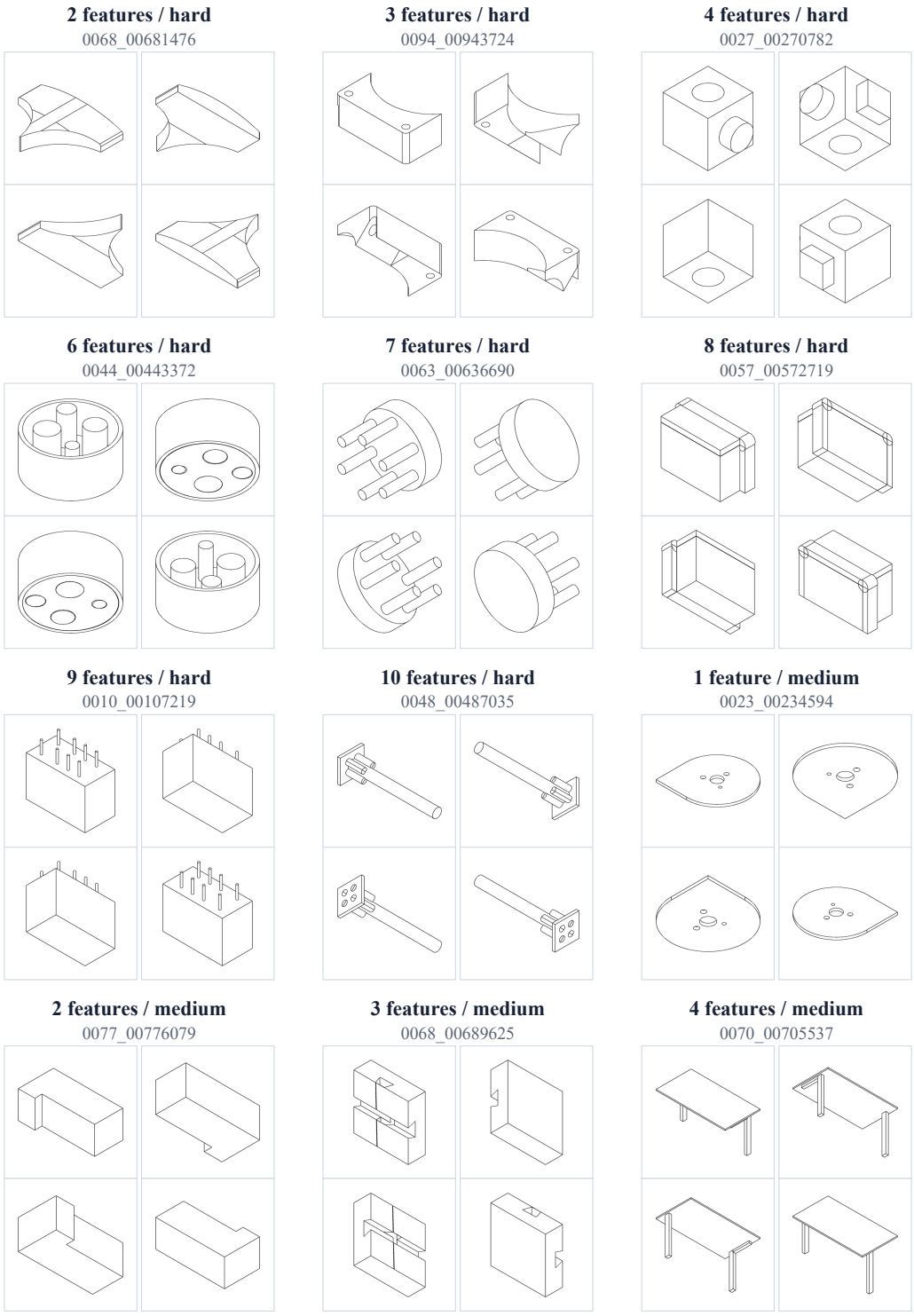


Figure 9: DeepCAD-Complex training examples (1/2), sourced from DeepCAD. Each card gives the feature count, structural-complexity tier, and source ID, followed by the same FRT/BRB/FLB/BLT views in a 2×2 arrangement. All drawings come from replaying the corresponding quantized ground-truth program.

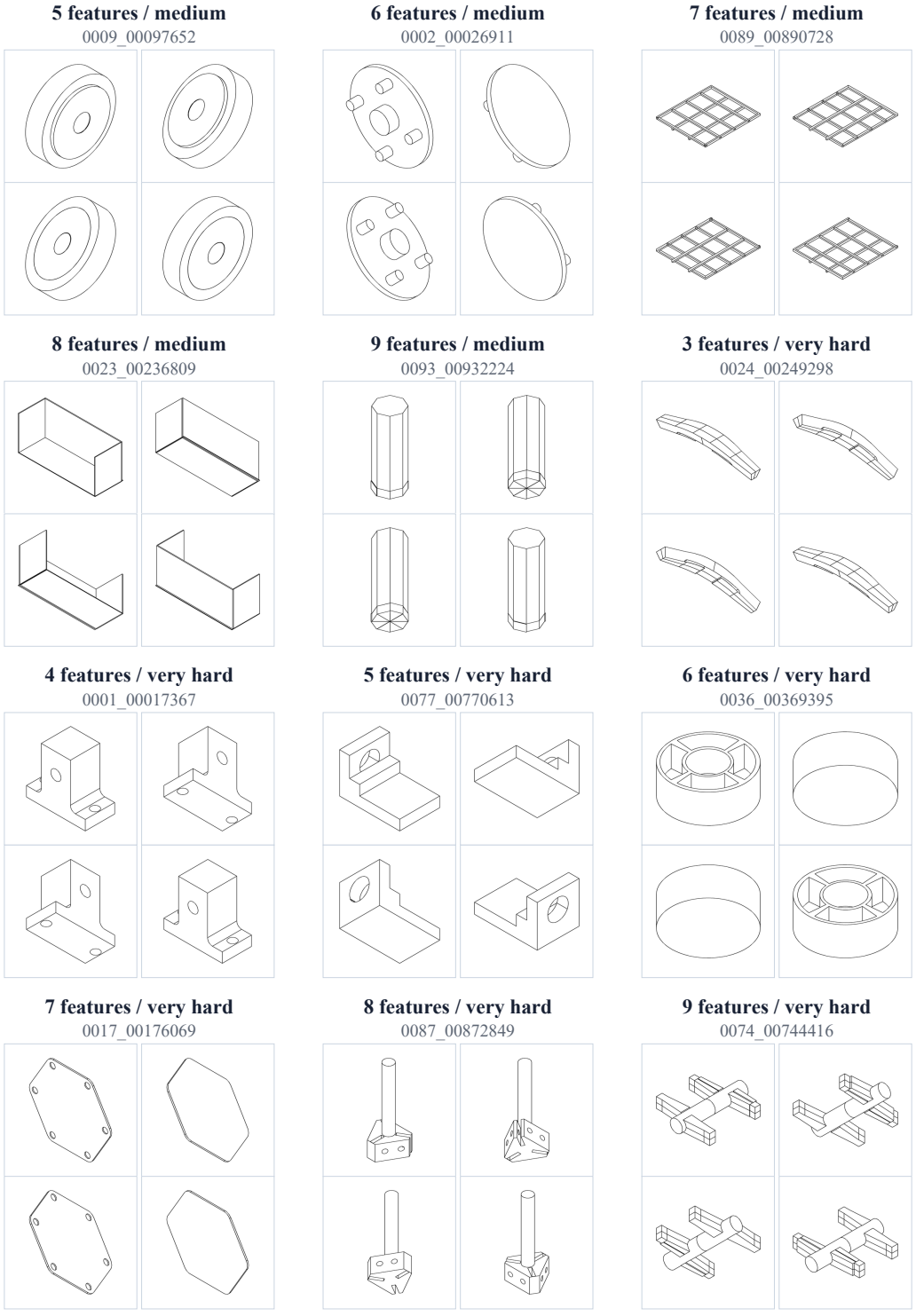


Figure 10: DeepCAD-Complex training examples (2/2), sourced from DeepCAD. The gallery illustrates varied profiles, construction lengths, and viewing ambiguities. Tier labels describe structural complexity, not reconstruction quality or empirical difficulty for a particular model.

C EXECUTION-DERIVED GEOMETRIC GROUNDING

This section expands the geometric objectives in Sec. 3.3. Appendix B describes how the supervision is obtained; here we specify how it trains the contextualized state and predictive transition representations.

C.1 GEOMETRIC FIELD REPRESENTATION

Fields at fixed spatial queries. For feature F_t, let $Q_t = \{q_{t,i}\}_{i=1}^{N}$ be its cached set of $N = 2{,}048$ spatial queries. Coordinates use the common program-level normalization in Eq. 14. We associate three conditional decoding tasks with each feature:

$$\begin{aligned}(H_t^b, y_{t,i}^b) &= (S_{t-1}, O_{t-1}(q_{t,i})),\\ (H_t^a, y_{t,i}^a) &= (S_t, O_t(q_{t,i})),\\ (H_t^p, y_{t,i}^p) &= (P_t, [A_t(q_{t,i}), R_t(q_{t,i})]).\end{aligned} \tag{18}$$

The state targets are binary scalars, whereas the transition target is a two-channel binary vector. Addition and removal denote the net material changes defined in Eq. 16, not the entire extrusion tool. Both transition channels remain supervised even when one is identically zero. The initial state has $O_0(q) = 0$.

The decoder predicts *field values at the supplied coordinates*, not point coordinates or a point-cloud correspondence. Queries may differ across features without changing the task definition. For a given feature, all three tasks share Q_t, which permits pointwise comparison of their outputs. Coordinates and noisy field values enter the auxiliary decoder; they are not supplied to the vision–language backbone.

Conditioning and causal information. Each H consists of four contextualized slot hidden states. S_{t-1} and P_t precede the content of F_t, while S_t follows it, as specified in Appendix A.1. Thus transition prediction must anticipate the next geometric change from the images and preceding program context, rather than read the ground-truth feature being predicted. The pre-state and post-state tasks share the same state decoder; the transition task shares its core blocks but has two-channel input and output projections (Appendix A.2). Sampling-group labels, component identifiers, and boundary-pair metadata are not decoder inputs.

C.2 CONDITIONAL FLOW MATCHING OBJECTIVE

Noisy interpolation and velocity regression. For task $j \in \{b, a, p\}$ with channel dimension $d_b = d_a = 1$ and $d_p = 2$, we construct the straight interpolation path

$$x_{t,i,\tau}^j = (1-\tau)\epsilon_{t,i}^j + \tau y_{t,i}^j, \qquad u_{t,i}^j = y_{t,i}^j - \epsilon_{t,i}^j, \tag{19}$$

where $\epsilon_{t,i}^j \sim \mathcal{N}(0, I_{d_j})$ and $\tau \sim \mathcal{U}(0,1)$. In each training forward pass, one time is sampled per feature and shared across its queries and three tasks; Gaussian noise is sampled independently across queries, channels, and tasks. Times and noise are resampled on subsequent visits, while the cached coordinates and ground-truth fields remain fixed.

C.3 STATE-TRANSITION CONSISTENCY CONSTRAINT

Clean-field estimates from the training path. The ground-truth fields satisfy $O_t = O_{t-1} + A_t - R_t$ at valid queries, but separately supervised predictions need not obey this relation. Using the same noisy inputs and velocities as the FM objectives, we form

$$\widetilde{y}_{t,i}^j = x_{t,i,\tau}^j + (1-\tau) v_\phi(q_{t,i}, x_{t,i,\tau}^j, \tau; H_t^j). \tag{20}$$

If the predicted velocity equals $y - \epsilon$, this estimate equals the clean target y. It is a single-step endpoint estimate on the training path, not a separately integrated ODE sample. We do not clamp or threshold these estimates during training.

Writing the three outputs as $\widetilde{O}_{t-1}$, $\widetilde{O}_t$, and $[\widetilde{A}_t, \widetilde{R}_t]$, the per-feature consistency loss is

$$\mathcal{L}_{\text{cons},t} = \sum_i w_{t,i}^{\text{cons}} \left[\widetilde{O}_t(q_{t,i}) - \widetilde{O}_{t-1}(q_{t,i}) - \widetilde{A}_t(q_{t,i}) + \widetilde{R}_t(q_{t,i})\right]^2. \tag{21}$$

It uses the same feature averaging as the FM terms. Gradients pass through all three decoding calls and their conditioning representations; none of the predicted fields is detached. CAD replay and occupancy classification are performed offline to create the targets, so this constraint requires neither a differentiable CAD kernel nor online execution during supervised training.

Role and scope of the constraint. For example, at a point removed by a cut, the target values are $(O_{t-1}, A_t, R_t, O_t) = (1, 0, 1, 0)$. Predicting removal while retaining the same post-state occupancy gives a nonzero consistency residual. The constraint therefore couples geometric predictions across the feature boundary, rather than imposing an arithmetic identity on the slot hidden states. It is a soft compatibility regularizer, not a guarantee of binary occupancies, disjoint predicted change regions, or executable CAD programs. In particular, constant zero fields would be consistent but geometrically incorrect, which is why consistency is used together with the individual FM objectives.

Combined supervised objective. Stages B and C use Eq. 9, with $\lambda_S = \lambda_P = 0.1$ and a target consistency coefficient of 0.01. The implementation ramps the latter over the first 100 updates of each stage: at one-based stage update k, $\lambda_{\text{cons}}(k) = 0.01 \min(k/100, 1)$.

D Representation-Guided Execution Alignment

This section details the fourth training stage (Stage D), introduced in Sec. 3.4. RGEA aligns the consequences of a generated feature with the geometric intent predicted by a frozen reference. It updates the language adapters and slot embeddings without differentiating through CAD execution or adding a geometric reconstruction loss to the policy update.

D.1 GRPO Optimization Details

Initialization and feature-level rollouts. Both the current policy π_θ and the frozen reference π_0 start from the completed Stage C checkpoint. Given the images and a ground-truth prefix $C_t = (I, F_{<t}^{\text{GT}})$, the policy produces four candidate continuations for F_t. The prefix contains the ordinary S/P slots at their feature boundaries, but excludes the current ground-truth feature content. Each candidate is executed independently from the same ground-truth pre-state G_{t-1}; candidates do not form a multi-step rollout chain. Sampling stops at the feature-end marker, with a limit of 2,048 generated DSL tokens. Incomplete or invalid features remain in the group as failed candidates rather than being resampled.

We use 3,000 non-first features from the supervised training collection, with 1,500 from each of Procedural Construction and DeepCAD-Complex. Each domain contributes 1,200 large-change and 300 medium-change features. Selection uses CAD-derived change volumes, not model rewards: relative change is $(V_{\text{add}}+V_{\text{remove}})/(V_{\text{before}}+V_{\text{add}})$, with large/medium bins defined by within-domain, within-operation 70th/40th percentiles. Both bins additionally pass the 20th-percentile normalized-change-volume threshold. At most three features are taken from any model or identical-DSL group. A separate 300-feature monitoring split, 150 per domain, is group-disjoint from Stage D training but originates from the Stage C training collection.

Relative advantages and policy loss. For the $K = 4$ rewards of a prefix, we compute

$$\bar{r} = \frac{1}{K}\sum_{i=1}^{K} r_i, \qquad \sigma_r = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(r_i - \bar{r})^2}, \qquad \widehat{\mathcal{A}}_i = \frac{r_i - \bar{r}}{\sigma_r}. \tag{22}$$

Groups with no executable candidate or $\sigma_r < 10^{-3}$ are skipped. In an active group, all four candidates, including failures, contribute to the update. There is no learned value model or replay buffer.

Let $a_{i,j}$ be token j of candidate i, L_i its token length, and $h_{i,j}$ its prefix context. The rollout distribution μ is the snapshot of the current policy used to sample that batch. Define

$$\rho_{i,j} = \frac{\pi_\theta(a_{i,j} \mid h_{i,j})}{\mu(a_{i,j} \mid h_{i,j})}, \qquad d_{i,j} = \log \pi_0(a_{i,j} \mid h_{i,j}) - \log \pi_\theta(a_{i,j} \mid h_{i,j}). \tag{23}$$

All three distributions use the same temperature and allowed action vocabulary. The implemented candidate loss is

$$\begin{aligned} \mathcal{L}_i = \frac{1}{L_i} \sum_{j=1}^{L_i} \Big[& - \min\Big(\rho_{i,j}\widehat{\mathcal{A}}_i, \mathrm{clip}(\rho_{i,j}, 1-\eta, 1+\eta)\widehat{\mathcal{A}}_i\Big) \\ & + \beta\rho_{i,j}\big(\exp(d_{i,j}) - d_{i,j} - 1\big)\Big], \end{aligned} \tag{24}$$

with $\eta = 0.2$ and $\beta = 0.02$. The second term is an importance-weighted sampled KL estimator relative to the fixed Stage C policy. Rollout/reference log probabilities and advantages are detached; gradients are retained through both current-policy factors in the KL term. Losses are averaged equally over candidates and active groups, after averaging tokens within each candidate. Image/prefix tokens, forced slots, and the transport end-of-sequence token are not policy actions.

Gradient path and update budget. Action probabilities are recomputed with gradients through the entire prefix, including the policy's own S/P slots. Consequently, the slot embeddings can change even though the reference representations used for reward computation remain frozen. The original backbone, vision encoder, vocabulary embeddings, output head, and geometric decoder are fixed. We process 16 prefix groups per batch, make one optimizer update per active batch, and traverse the data once. A fully inactive batch advances the data cursor without an update; it is not replaced with additional data.

Table 9: Feature-level RGEA settings. The four GPUs have distinct roles: two policy-gradient workers, one single-GPU vLLM rollout worker, and one frozen-reference worker. CAD execution runs on CPUs.

Setting	Value
Candidate group / prefix groups per batch	4 / 16
Sampling temperature / top-p	0.8 / 1.0
Top-k filtering / repetition penalty	Disabled / 1.0
Feature-token limit / context limit	2,048 / 16,384
Optimizer / weight decay	AdamW / 0
LoRA / slot learning rate	5×10^{-6} / 10^{-5}
Gradient-norm clipping	1.0
GRPO clipping / KL coefficient	0.2 / 0.02
Reward reliability floor δ	0.25
Field decoding: Euler steps / trajectories	16 / 2

D.2 Reward Design

Frozen geometric intent. The reference processes C_t to obtain S^0_{t-1} and P^0_t. Its frozen decoder predicts pre-state occupancy and added/removed material fields on a shared query set. We write these predictions with hats to distinguish them from CAD-derived ground truth:

$$p(q) = \mathrm{clip}\left(\widehat{O}^0_{t-1}(q) + \widehat{A}^0_t(q) - \widehat{R}^0_t(q), 0, 1\right). \tag{25}$$

Thus the pre-state term in the reward target is also decoded from S; it is not replaced by ground-truth occupancy. No post-feature state slot or ground-truth endpoint is used to produce p. Each field is decoded from Gaussian noise using 16 Euler steps, clamped to $[0, 1]$, and averaged over two trajectories before forming the clipped post-state. Noise seeds are fixed by feature identity, and the same p is used for all candidates in the group.

One shared spatial measure. The original 2,048 cached queries carry 75% of the base weight, following their normalized consistency-loss weights. We add 512 candidate-aware coverage queries

carrying the remaining 25% uniformly: 256 are uniform in a box covering the original query domain, GT post-state, and executable candidate post-states, with a 2% margin; 256 are surface samples distributed across these post-state solids and perturbed with normalized-coordinate Gaussian noise of standard deviation 0.002. These additional queries can expose unintended material outside the GT change region. All candidates are scored on this same 2,560-point set, rather than on separately chosen favorable point sets.

Let $y(q)$ be GT post-state occupancy, $x_i(q)$ candidate post-state occupancy, $w_0(q)$ the base weight, and $m(q)$ the common validity mask for the GT post-state and all executable candidates. The normalized reliability-calibrated weight is

$$\begin{aligned} c(q) &= \delta + (1-\delta)\big(1 - |p(q) - y(q)|\big)^2, \\ w(q) &= \frac{w_0(q)m(q)c(q)}{\sum_{q'} w_0(q')m(q')c(q')}, \qquad \delta = 0.25. \end{aligned} \tag{26}$$

The floor prevents low-confidence but valid regions from being assigned zero weight. Ground truth calibrates the reliability of the reference prediction; the candidate is then compared with that prediction:

$$r_i = \begin{cases} 1 - \sum_q w(q)|x_i(q) - p(q)|, & \text{if candidate } i \text{ is executable,} \\ -1, & \text{otherwise.} \end{cases} \tag{27}$$

An executable candidate has reward in $[0, 1]$.

E Baseline and Evaluation Details

E.1 Baseline Implementation

Paper-based adaptations on a common backbone. The CAD-method baselines in Table 1 are our *adapted reimplementations*, rather than evaluations of the authors' original checkpoints. Since usable official code or pretrained weights were unavailable to us for these experiments, we implemented the methods according to their paper descriptions on Qwen3-VL-4B-Instruct, the same backbone used by ForeCAD. These adaptations use the FeatureState-CAD training data, four-view inputs, and common executable CAD representation, with matched training amounts. This protocol compares the transferred method designs within a shared image-to-program interface; it does not claim to reproduce the original architectures or published benchmark numbers exactly.

This adaptation protocol applies to CAD2Program (Wang et al., 2025b), CAD-GPT (Wang et al., 2025a), PlankAssembly (Hu et al., 2023), A3PL (Guan & Sun, 2026), SOV-CAD (Feng et al., 2026), and GIFT (Giannone et al., 2026). Their names identify the source method being adapted, not a claim that its original backbone or output parameterization is unchanged. In particular, matching the final DSL interface does not by itself specify how each method's internal representation and auxiliary supervision are transferred.

Sequence-only controls and model identities. The Qwen3-VL-4B row in Table 1 is a sequence-only supervised control with the matched training amount described in Sec. 5.1: language LoRA is trained with full-program CAD CE, without S/P slots, geometric-field supervision, or RGEA. Its optimizer configuration follows Appendix A.4. By contrast, the all-disabled row in Table 2 uses the Stage A endpoint described in Appendix A.3. These two rows correspond to different training budgets. The InternVL3-8B row uses the actual InternVL3-8B backbone, rather than a Qwen3-VL-based adaptation.

E.2 Evaluation Protocol

Full-program generation. Evaluation starts from the four input images and the reconstruction instruction, with no GT program prefix or GT geometry supplied to the generator. The ForeCAD checkpoints and Qwen sequence-only control use one greedy output per input (temperature 0, seed 42), with a limit of 4,096 generated DSL tokens and a 32,768-position context limit. The DSL budget excludes internal S/P positions and the transport end-of-sequence token. ForeCAD inserts its four state and four transition slots at the boundaries described in Appendix A.1; these positions

remain in the model context but are removed from the executable output. Generation ends at the program boundary or the token limit. This is free-running generation, not teacher-forced scoring of the GT sequence.

Strict parsing and execution rate. The evaluator parses each emitted DSL directly, without extracting code from Markdown, repairing syntax, deleting failed features, or substituting a successfully executed prefix. It replays the parsed program feature by feature with the same CAD execution backend used for the dataset. A successful reconstruction requires every intermediate and final state to pass the CAD-kernel validity check, contain exactly one solid, have finite volume greater than 10^{-12}, and have finite bounding-box extents greater than 10^{-9} along all axes. These tolerances are in the replay coordinate system. The same geometric validity checks apply to GT programs.

For a setting containing N inputs, let e_i indicate successful parsing and execution of prediction i. We report

$$\text{Exec.} = \frac{100}{N} \sum_{i=1}^{N} e_i. \tag{28}$$

Execution rate does not require exact text, an identical feature count, or geometric agreement with the GT. Completion status, parse success, and feature-count agreement are recorded separately. Capped or failed outputs are not removed from the evaluation population.

Volumetric IoU. For executable predicted and GT solids $\widehat{G}_i$ and G_i, the per-object score is

$$J_i = \frac{\text{Vol}(\widehat{G}_i \cap G_i)}{\text{Vol}(\widehat{G}_i) + \text{Vol}(G_i) - \text{Vol}(\widehat{G}_i \cap G_i)}. \tag{29}$$

The intersection is obtained by a B-rep Boolean operation and volumes are computed by the CAD kernel. This metric is not estimated from the training query points or a voxel grid. Small numerical excesses in intersection volume are clamped to the smaller operand volume; an excess above 10^{-6} times the larger operand volume is flagged as a metric error. No occupancy approximation is silently substituted. The reported IoU is $100N^{-1}\sum_i \widetilde{J}_i$, where $\widetilde{J}_i = J_i$ when available and is zero for failed execution or an unavailable IoU. Thus IoU uses the complete population, rather than only successfully reconstructed objects. Metric failures remain explicit in the evaluation records.

Surface Chamfer distance. For each executable pair, we sample 4,096 points from each surface. The solids are tessellated with linear tolerance 0.001 and angular tolerance 0.1; nondegenerate triangles are sampled proportionally to area, followed by uniform barycentric sampling within a triangle. Sampling is deterministic given a seed derived from the sample identity. Both point sets are normalized by the *same* GT metadata: $\widetilde{x} = (x - c_i)/\ell_i$, where c_i and ℓ_i are the stored GT normalization center and cube extent. There is no separate prediction rescaling, rigid registration, or ICP alignment.

Let U_i and V_i denote the normalized predicted and GT surface point sets. We compute

$$\text{CD}_i = \frac{1}{2}\left[\frac{1}{|U_i|}\sum_{u\in U_i}\min_{v\in V_i}\|u-v\|_2 + \frac{1}{|V_i|}\sum_{v\in V_i}\min_{u\in U_i}\|v-u\|_2\right]. \tag{30}$$

Distances are Euclidean and *not squared*. CD is dimensionless after GT normalization, with no factor of 10^3 or percentage conversion. CD_{mean} and CD_{med} are the mean and median of the per-object CD_i values over finite, successfully computed distances. The median is not taken over the individual nearest-neighbor distances. Missing distances are not replaced by zero. Accordingly, CD is conditional on each model's valid subset and must be interpreted alongside execution rate and the number of valid CD measurements. Tables round CD to three decimal places only after aggregation.

E.3 Additional Quantitative Results

Feature-level intervention diagnostics. Figure 4 concerns the final Stage D model and a single next feature under a fixed GT prefix; it is not another full-program reconstruction benchmark. The conditions retain the original representations, replace P with a mean representation, or replace P or

S with a matched cross-example representation. Mean-P replacement does not mean removing the slots or applying an attention mask. For a fixed pre-state, execution of the generated feature gives binary added/removed fields $\Delta^{\text{exec}}(q) = (A^{\text{exec}}(q), R^{\text{exec}}(q))$. GT replay similarly gives $\Delta^{\text{GT}}(q)$, and the geometric decoder gives $\widehat{\Delta}^{P}(q)$ from the supplied transition representation, thresholded at 0.5.

On the common valid transition-query set $\mathcal{Q}$, define

$$\mathcal{J}_{\Delta}(B, C) = \frac{\sum_{q \in \mathcal{Q}} \sum_{k \in \{\text{add},\text{remove}\}} \mathbf{1}[B_k(q) = 1 \ \wedge \ C_k(q) = 1]}{\sum_{q \in \mathcal{Q}} \sum_{k \in \{\text{add},\text{remove}\}} \mathbf{1}[B_k(q) = 1 \ \vee \ C_k(q) = 1]}. \tag{31}$$

This is a micro-IoU over signed query–channel pairs, with zero assigned to an empty union. GT-ΔIoU is $\mathcal{J}_{\Delta}(\Delta^{\text{exec}}, \Delta^{\text{GT}})$; P$\rightarrow$Exec is $\mathcal{J}_{\Delta}(\widehat{\Delta}^{P}, \Delta^{\text{exec}})$. The latter uses donor P for the matched P replacement condition and the original P otherwise. It is unreported for mean-P replacement, which does not provide an instance-specific intent target. These are unweighted IoUs on GT-change-balanced spatial queries, not the volumetric IoU in Eq. 29 or the soft RGEA reward. Model parse/execution failures receive zero; infrastructure failures abort scoring. Features are averaged within each object before averaging objects equally within a setting.

Latency measurement. Figure 5 compares the Stage A SFT endpoint, the slots-only Stage C endpoint, and the final Stage D model. We use the same input order across models, three separate warm-up objects per setting, and one measured pass per model. Models run sequentially with one active request, tensor parallelism 1, bfloat16, greedy decoding, and prefix and multimodal-processor caches disabled. The instance reports an A100-SXM4-80GB GPU with a 20 GiB HAMI-managed allocation; whole-card compute exclusivity is not established.

We time a request from enqueueing through its final returned output. This includes model-side prefill, autoregressive decoding, internal slot processing, and the streaming observation path, but excludes model loading, warm-up, caller-side image preparation, output-file writing, and CAD execution/scoring. Full-program means include all 100 inputs in each setting, including outputs reaching the 4,096-token limit. Different models can produce different program lengths, so shorter full-program latency alone does not establish lower per-token cost or better reconstruction.

Matched-length measurements are directly observed times to reach 128, 256, or 512 DSL tokens during the same free-generation runs. They are not total times divided by sequence length and rescaled. For each length we retain only inputs reaching that exact token boundary in all three models, with equal input-token counts and no generation errors. The common populations contain 200, 190, and 74 objects, respectively. Slots do not count toward these DSL lengths, but their elapsed computation is included.

F Additional Qualitative Results

We examine saved, free-running outputs of the final Stage D model from both FeatureState-CAD settings. Each row shows the original four input views, GT, and the execution of ForeCAD's unmodified program. GT and prediction share the same camera, coordinate frame, and scale, without independent alignment or rescaling.

F.1 Successful Reconstruction Examples

Procedural Construction. Figure 11 shows four mechanical structures. ForeCAD preserves the slider's elongated groove and small openings, the pulley's concentric recess and central bore, the split clamp's transverse bore and top slot, and the channel mount's upright walls and base. These examples span three to six GT features and require compatible sketches, extrusion parameters, and Boolean operations. The executable programs retain the main spatial relationships rather than only the outer silhouette.

DeepCAD-Complex. Figure 12 shows a stepped bracket, an open channel, a recessed cylindrical body, and a plate with a hollow boss. The dominant structures are recovered, but local differences remain; successful reconstruction denotes executable, geometrically close output, not identical parameters or construction histories. For example, panel (d) uses three predicted features versus two

GT features and achieves 85.9% IoU, illustrating why we evaluate executed solids rather than require identical DSL strings.

F.2 Failure Case Analysis

Figure 13 separates geometric errors in executable programs from construction failures.

Parameter errors despite a plausible shape. In panel (a), the predicted bushing remains executable and retains the two-feature construction and Boolean-operation types. However, the first sketch-size value changes from 66 to 77 and the second from 111 to 116, together with shifts in the sketch origins. The result has altered bore/body dimensions and only 70.8% IoU. Panel (b) shows a larger discrepancy: the generated cylindrical part has excessive height and incorrect wall proportions, giving 13.9% IoU despite successful execution. These examples distinguish recognizing an overall shape from recovering its numerical geometry.

Missing structure in a completed program. In panel (c), ForeCAD emits a complete three-feature program for a five-feature GT object. The executed result reduces the paired-rail structure to a much smaller, simpler solid and achieves 7.8% IoU. This is not token-limit truncation: the saved output reaches the program-end marker. Nor does the lower feature count alone define the error; the failure is the missing geometry in the executed result.

Syntactic completion without geometric validity. Panel (d) is complete and parseable, but feature 3 produces an invalid solid despite a solid count of one. We mark this failure explicitly, without repairing the output or substituting an executable prefix. Correct DSL syntax and termination do not guarantee geometric validity.

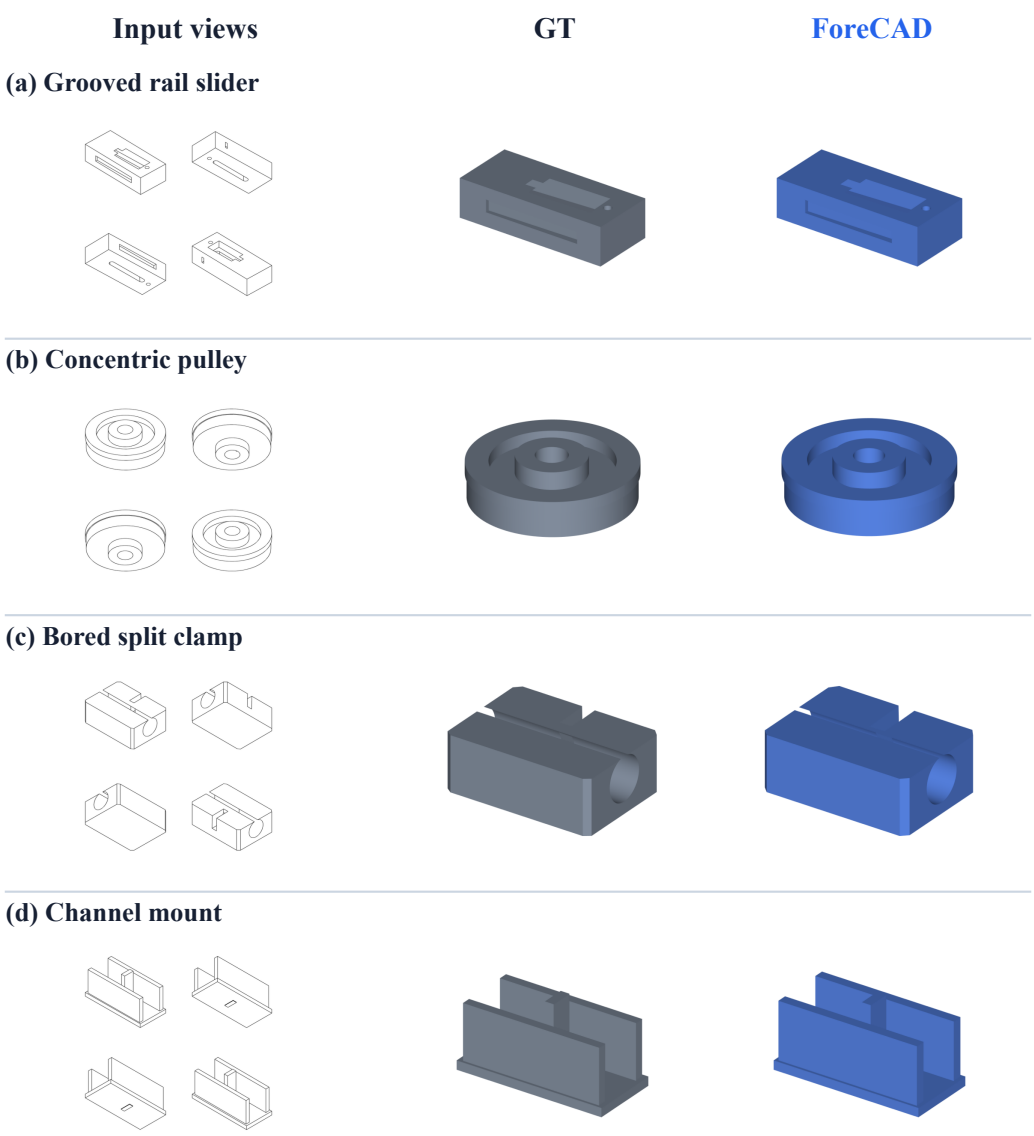


Figure 11: **ForeCAD reconstruction examples on Procedural Construction.** Each row contains the four input views, GT (gray), and the executed ForeCAD prediction (blue). Input views follow the dataset's fixed order in a row-major 2×2 grid. The same camera and world-coordinate scale are used for each GT/prediction pair. These are selected qualitative examples, not an aggregate comparison.

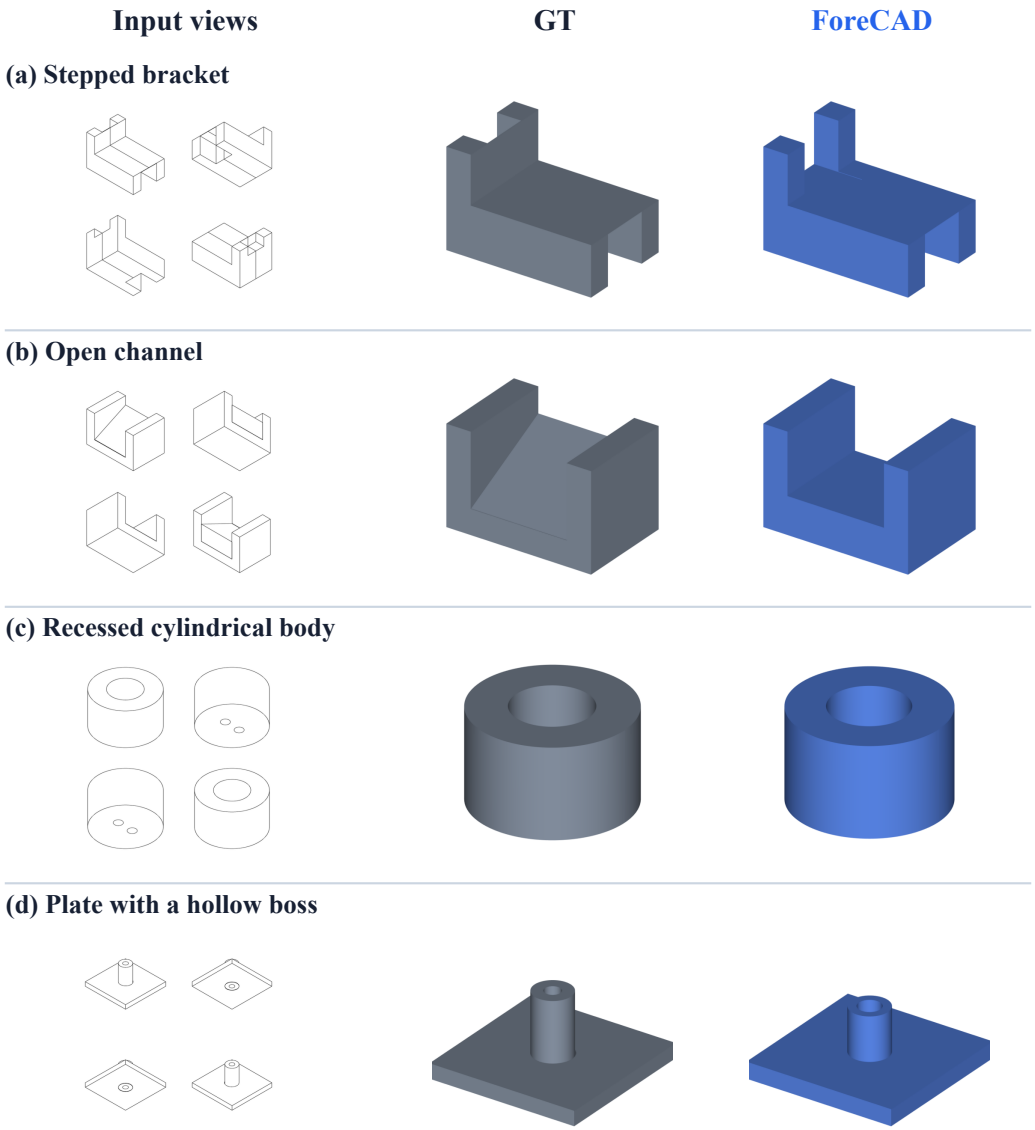


Figure 12: **ForeCAD reconstruction examples on DeepCAD-Complex.** The layout and rendering conventions match Figure 11. The examples illustrate recovery of stepped, open-channel, recessed, and raised-boss structures. High geometric agreement does not imply identical CAD histories or perfect recovery of local dimensions.

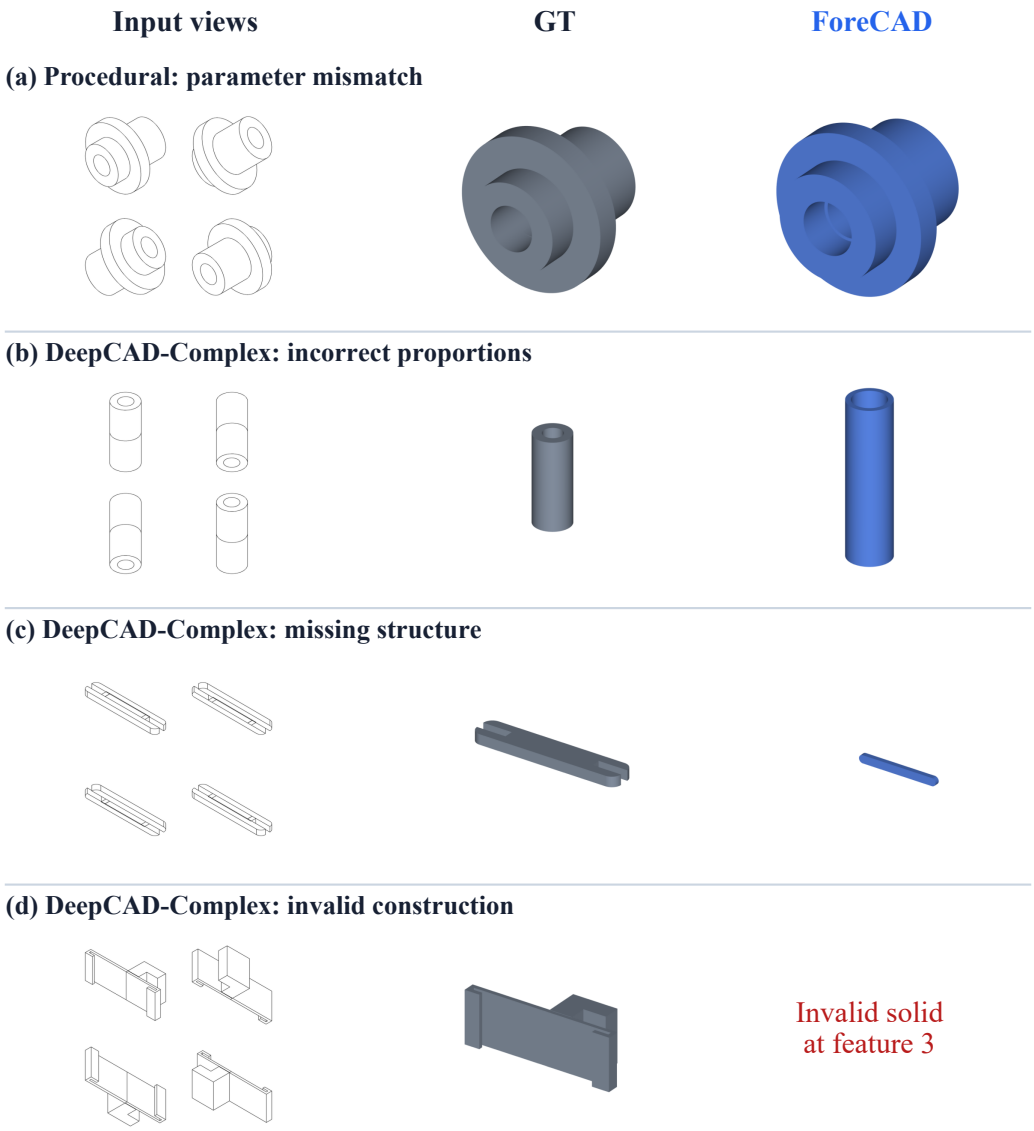


Figure 13: **Selected failure cases of ForeCAD.** (a) A plausible but incorrectly parameterized Procedural object. (b) Incorrect proportions in a DeepCAD-Complex object. (c) Missing structure in a completed, executable program. (d) A parseable program whose third feature produces an invalid solid. The first three rows show actual executed predictions, not repaired geometry. The last row is explicitly marked as failed and has no rendered final prediction. Pairwise shared scale preserves the size differences in (a)–(c).