CAD-WAM: ANTICIPATING GEOMETRY FOR TEXT-TO-CAD GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating editable computer-aided design (CAD) models from text requires selecting operations that produce the intended changes to the geometry constructed so far. An incorrect operation can compromise subsequent steps, making reliable multi-step generation challenging. Existing methods based on large language models (LLMs) primarily learn to predict commands, without explicit supervision of the geometric states those commands should produce. Inspired by world-action models, we introduce CAD-WAM, which couples geometry-conditioned action generation with next-state prediction. We define complete modeling operations as actions in a CAD domain-specific language (DSL) and encode surface point clouds of intermediate solids into compact latent states using a pretrained variational autoencoder. An LLM backbone generates the next action from the target description and current geometric state, while an auxiliary head on the same backbone predicts the next state to provide geometric supervision during training. We construct step-wise training trajectories from parametric CAD models in the DeepCAD dataset, pairing each trajectory with an expert-level textual description. At inference, each generated action is executed by a CAD kernel, and the resulting solid is re-encoded to inform the next decision. On the DeepCAD test set, CAD-WAM outperforms the task-specific baselines in geometric accuracy and output validity, with larger gains on models requiring more operations, and produces more accurate geometry than general-purpose LLMs prompted with the same DSL. Ablations further show that next-state supervision reduces mean Chamfer distance by 13.2% relative to an otherwise identical model trained only to predict actions.

1 INTRODUCTION

Parametric computer-aided design (CAD) represents a model through a sequence of editable geometric operations rather than only its final shape. Generating such models from natural language can reduce the manual effort required for CAD modeling. Existing approaches generate parametric command sequences (Khan et al., 2024) or fine-tune large language models (LLMs) to produce executable programs in CAD scripting languages such as CadQuery and FeatureScript (Guan et al., 2025; Pyatov et al., 2026; Kolodiazhnyi et al., 2026).

Reliable multi-step CAD generation requires more than producing executable commands: each operation must also realize the intended change to the geometry constructed so far. An incorrect operation can alter the solid on which subsequent operations act, causing further deviations from the target. Our experiments also reveal an uneven error distribution: several baselines achieve low median geometric error, yet some outputs deviate substantially from the target (Figure 4). Existing methods incorporate geometric information through reinforcement learning with geometric rewards (Guan et al., 2025; Kolodiazhnyi et al., 2026) or step-wise generation conditioned on the current boundary representation (B-rep) (Qi et al., 2026). These approaches use geometry to evaluate generated programs or inform the next decision, but do not explicitly supervise predictions of the next geometric state. This motivates learning each modeling step through both its symbolic action and its intended geometric outcome, connecting operation generation with the evolving shape of the CAD model.

World-action models provide a natural perspective on this formulation by coupling action learning with prediction of future observations (Tian et al., 2025; Zhu et al., 2025). Future prediction can pro-

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.

vide additional supervision during training without making action generation depend on predicted states at inference (Ye et al., 2026). CAD offers a direct source of such supervision: executing expert programs yields the intermediate solids after each operation. These geometric outcomes can supervise the representation used for action generation, while the CAD kernel provides the actual geometry for subsequent decisions during inference.

Building on this idea, we introduce CAD-WAM, which jointly learns next-action generation and next-state prediction from the target description and current geometry. We design a CAD domain-specific language (DSL) that groups a sketch, its extrusion, and the associated Boolean operation into a complete modeling action, yielding a solid after each successful execution. We pretrain Set-VAE (Kim et al., 2021), a variational autoencoder for point clouds, on surface point clouds of intermediate and final solids from the training split. Its encoder is then frozen to provide compact latent states that serve both as geometric inputs and as targets for next-state prediction. The Geometric State Embedding maps the current state to geometric tokens, which are processed together with textual tokens by a shared LLM backbone. The Action Head generates the next DSL action, while the State Prediction Head regresses the next latent state. Joint training allows next-state supervision to shape the shared representation used for action generation. At inference, each generated action is executed by a CAD kernel, and the resulting solid is re-encoded to condition the next decision.

To train CAD-WAM, we construct step-wise trajectories from DeepCAD, a dataset of parametric CAD models (Wu et al., 2021), by converting their construction sequences into our DSL and executing them operation by operation. Each trajectory is paired with an expert-level textual description released with Pointer-CAD (Qi et al., 2026). At each step, the description and current geometric state form the input, while the next action and resulting geometric state provide supervision. We compare CAD-WAM with task-specific baselines and general-purpose LLMs on the DeepCAD test set, applying a common geometric evaluation protocol to all available outputs. CAD-WAM achieves higher geometric accuracy and lower output invalidity than the evaluated task-specific baselines, with larger gains on models requiring more operations. It also produces more accurate geometry than general-purpose LLMs prompted with the same DSL. Ablations show that the latent-state model outperforms its textual-state counterpart and that next-state supervision reduces mean Chamfer distance by 13.2% relative to an otherwise identical model trained only to predict actions.

Our contributions are summarized as follows:

- We introduce CAD-WAM, a text-to-CAD framework that couples geometry-conditioned action generation with next-state prediction, while using kernel-executed geometry as feedback during inference.
- We develop an operation-level CAD DSL and a compact latent state representation, and construct step-wise training data that pair executable modeling actions with their geometric outcomes.
- We demonstrate improved geometric accuracy over the evaluated task-specific and general-purpose models on the DeepCAD dataset, and establish the benefit of next-state supervision through controlled ablations.

2 Related Work

2.1 CAD Generation with Language Models

Learning-based CAD generation is usually formulated over the construction sequence rather than the final geometry. DeepCAD introduced this formulation together with a dataset of 178,238 sketch-and-extrude sequences (Wu et al., 2021), and later models improved generation and controllability over the same representation without language supervision (Xu et al., 2022; 2023). Language models then replaced the task-specific command vocabulary by executable code: CAD-Recode reverse engineers CadQuery programs from point clouds (Rukhovich et al., 2025), and cadrille extends this setting to point clouds, images and text with reinforcement learning (Kolodiazhnyi et al., 2026). For text-conditioned generation, models emit either command sequences (Khan et al., 2024; Govindarajan et al., 2026) or programs in CadQuery (Guan et al., 2025) and FeatureScript (Pyatov et al., 2026). Closest to our setting, Pointer-CAD generates a model step by step and conditions each step on the boundary representation obtained so far (Qi et al., 2026). Unlike these methods, which either

ignore the intermediate geometry or merely condition on it, our work encodes it as a latent state and predicts the state that the next action produces.

2.2 World Models and World-Action Models

World models learn the dynamics of an environment in a latent space, so that the effect of an action can be estimated without executing it (Ha & Schmidhuber, 2018; Hafner et al., 2020). The idea has since been scaled to generative video models, which synthesize future observations conditioned on text or on latent actions (Du et al., 2023; Bruce et al., 2024).

Predicting future observations is useful for control only when it informs the choice of action. World-action models therefore couple prediction with action generation, and differ in how the two are combined: the action may be inferred from a predicted future observation (Tian et al., 2025), actions and observations may be produced within a single generative process (Zhu et al., 2025), or future prediction may serve as an auxiliary objective during training (Ye et al., 2026). CAD-WAM draws on this idea and applies it to CAD modeling, where the transition induced by an action is deterministic and the resulting geometry can be obtained exactly with a CAD kernel. The predicted state is therefore a well-defined target, and the observed state requires no approximation.

3 CAD-WAM

3.1 Problem Formulation

Given a natural-language description g of a target CAD model, CAD-WAM constructs the model in a step-by-step manner through a sequence of state-dependent decisions. We design a CAD domain-specific language (DSL; Section 3.2) that represents CAD operations (e.g., extrusions and Boolean operations) as executable actions. The construction process is thus expressed as an action sequence $(a_0, a_1, \ldots, a_{T-1})$, where $a_t \in \mathcal{A}$ and $\mathcal{A}$ denotes the action space defined by the DSL. Let S_t denote the B-rep of the intermediate CAD model after t actions. Each action updates the current geometry through a deterministic transition implemented by the CAD kernel:

$$S_{t+1} = \mathcal{K}(S_t, a_t), \qquad t = 0, \ldots, T-1, \tag{1}$$

where $\mathcal{K}$ denotes the CAD kernel's execution function.

We represent each nonempty intermediate model by a compact geometric state. Specifically, we sample a point cloud $\mathcal{P}(S_t)$ from its surface and encode it using a SetVAE encoder E_ϕ (Kim et al., 2021) (Section 3.3):

$$\mathbf{z}_t = E_\phi\big(\mathcal{P}(S_t)\big) \in \mathbb{R}^{1024}. \tag{2}$$

The process starts from an empty model S_0, represented by the zero vector $\mathbf{z}_0 = \mathbf{0}$.

CAD-WAM conditions each decision on the textual target and the current geometric state, without explicitly taking the action history as input. Given $(g, \mathbf{z}_t)$, it jointly predicts the next action and the resulting geometric state by modeling

$$p_\theta\big(a_t, \mathbf{z}_{t+1} \mid g, \mathbf{z}_t\big), \tag{3}$$

where θ denotes the parameters of CAD-WAM. This formulation couples action generation with prediction of the resulting geometric state, encouraging the model to anticipate the geometric consequences of the next action.

3.2 CAD Domain-Specific Language

We design a CAD DSL in which each modeling action produces a CAD solid upon successful execution, providing an explicit geometric state after each modeling step. Figure 1 shows a construction sequence in this DSL, the JSON form of an action, and the procedure by which an action is executed. In contrast, a single command token (Wu et al., 2021) or code statement (Rukhovich et al., 2025; Guan et al., 2025) does not necessarily yield a solid, which leaves the intermediate geometry difficult to represent. The DSL comprises a geometric layer, which defines planar profiles on sketch planes as closed loops of lines, arcs, and circles with nested loops forming holes, and an action layer

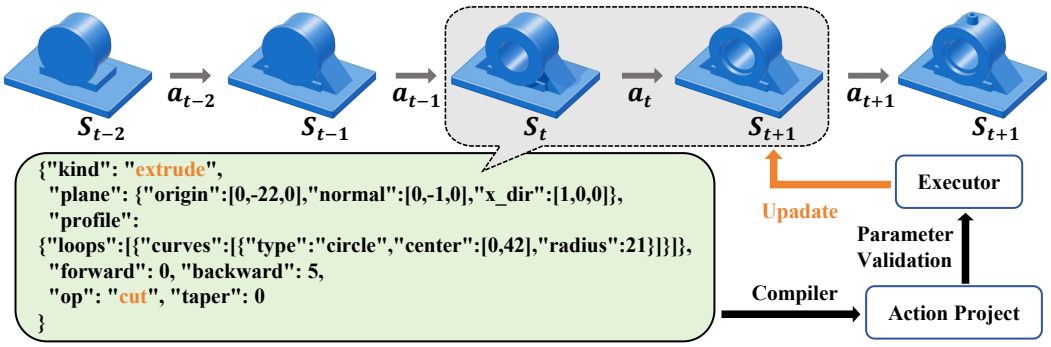


Figure 1: Our CAD DSL. Top: a construction sequence in which every action yields a valid solid. Bottom left: an action serialized as a JSON object. Bottom right: execution of an action, whose parameters are validated before the CAD kernel is invoked.

built on these profiles. Each action is a one-sided, symmetric, or two-sided sketch-based extrusion, with its Boolean operation on the current model as a parameter.

Actions are serialized as JSON objects with continuous parameters, and a dedicated end action terminates the construction. We also check the parameters of each action before passing it to the CAD kernel (CadQuery contributors, 2026). The complete grammar and validation rules are provided in Appendix A.

3.3 Geometric State Representation

World models commonly encode high-dimensional observations into a compact latent space with a variational autoencoder (VAE) and model dynamics within that space (Ha & Schmidhuber, 2018; Hafner et al., 2020). Inspired by this practice, we represent each intermediate model S_t by a latent geometric state. We observe S_t through a point cloud sampled from its surface, which captures the shape of the solid in a unified form for models of arbitrary complexity. We encode this point cloud with SetVAE (Kim et al., 2021), a permutation-invariant VAE for sets, into a compact fixed-dimensional state. Its KL-regularized latent space provides a smooth target for state prediction.

To construct $\mathcal{P}(S_t)$, we sample 2,048 points uniformly over the surface of S_t. We then center them at the bounding-box center and scale them to a fixed extent, so that intermediate solids of very different sizes are encoded at a comparable scale. Since this normalization removes absolute position and scale, we supply the size and center of the bounding box of S_t to the model as part of its textual input. SetVAE encodes a set through a hierarchy of seven latent levels, which contain 1, 1, 2, 4, 8, 16, and 32 latent vectors of dimension 16. We concatenate the posterior means of all levels to obtain the deterministic geometric state $\mathbf{z}_t \in \mathbb{R}^{1024}$. We pretrain SetVAE on the point clouds of all intermediate and final solids in the training split (Section 4) and use its encoder as E_ϕ, so that the encoder covers the partial models encountered as states. Training details and reconstruction quality are reported in Appendix B.

3.4 Model Architecture

Figure 2 illustrates the architecture of CAD-WAM, which realizes the joint distribution in Eq. (3) with a shared LLM backbone and two prediction branches. At step t, the Geometric State Embedding converts the geometric state $\mathbf{z}_t$ into $K = 16$ geometric tokens. These tokens are integrated with the textual tokens and jointly processed by the backbone. Its hidden states are then decoded along two branches: the Action Head generates the next action a_t, and the State Prediction Head regresses the next geometric state $\hat{\mathbf{z}}_{t+1}$. In addition to the pretrained LLM, CAD-WAM introduces only the Geometric State Embedding and the State Prediction Head, which together contain 11.4M parameters, fewer than 1% of a 1.5B-parameter backbone.

Geometric State Embedding. Multimodal LLMs commonly expose a continuous representation to the backbone as a short sequence of tokens (Alayrac et al., 2022; Li et al., 2023). Following

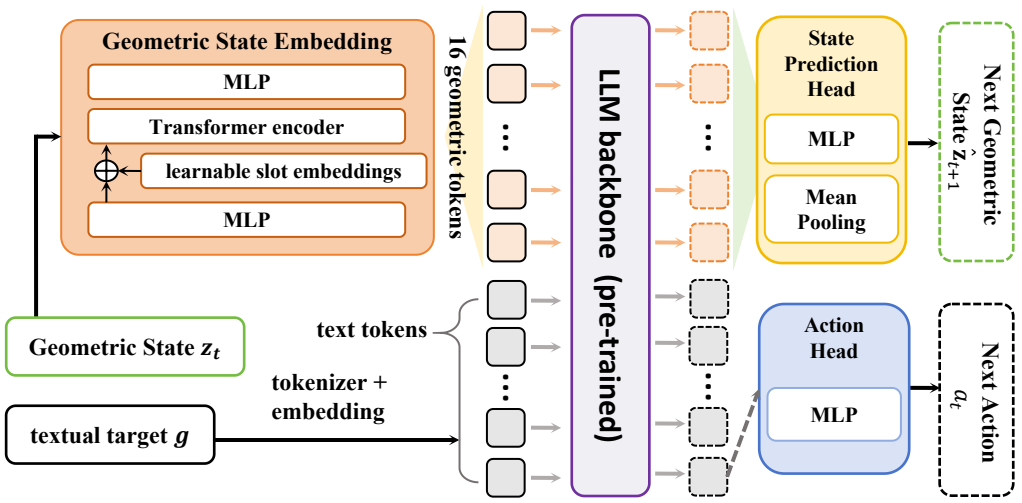


Figure 2: Architecture of CAD-WAM: the Geometric State Embedding maps the geometric state to 16 geometric tokens that follow the textual tokens in the backbone; the Action Head generates the next action and the State Prediction Head regresses the next geometric state.

this practice, we design the Geometric State Embedding, which maps the geometric state $\mathbf{z}_t$ to K geometric tokens in the embedding space of the backbone. The module first normalizes $\mathbf{z}_t$ and expands it with $\mathbf{W}_1$ into K slots of width d_g, to each of which a learnable slot embedding in $\mathbf{P}$ is added. A two-layer bidirectional Transformer encoder f_g then allows the slots to exchange information, after which the projection $\mathbf{W}_2$ maps them to the hidden size d of the backbone:

$$\mathbf{G}_t = \mathrm{LN}\Big(f_g\big(\,\mathrm{reshape}(\mathbf{W}_1\,\mathrm{LN}(\mathbf{z}_t)) + \mathbf{P}\big)\Big)\mathbf{W}_2 \in \mathbb{R}^{K\times d}, \tag{4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{Kd_g\times 1024}$, $\mathbf{P} \in \mathbb{R}^{K\times d_g}$, $\mathbf{W}_2 \in \mathbb{R}^{d_g\times d}$, LN denotes layer normalization, and biases are omitted for brevity.

Shared Backbone and Action Head. At each step, the token embeddings of the textual input, which contains the textual target g, are followed by the K geometric tokens $\mathbf{G}_t$, and together they form the input sequence of the backbone. Placing the geometric tokens after the text allows their hidden states to attend to the entire textual input under causal attention. The resulting sequence is processed by a pretrained decoder-only LLM, which we leave architecturally unchanged. The Action Head is the pretrained language-modeling head of this backbone, and it generates the next action a_t autoregressively as text in our DSL. Since actions are expressed as text, this head can be reused directly, thereby preserving the generative capability acquired during pretraining.

State Prediction Head. We design the State Prediction Head to predict the geometric state that the next action will produce. Let $\mathbf{H}_t \in \mathbb{R}^{K\times d}$ denote the final hidden states at the K geometric positions, which encode the textual target together with the current geometric state. The head averages these hidden states, normalizes the result, and regresses the next geometric state through a two-layer MLP with a GELU activation:

$$\hat{\mathbf{z}}_{t+1} = \mathrm{MLP}\left(\mathrm{LN}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{H}_{t,k}\right)\right) \in \mathbb{R}^{1024}. \tag{5}$$

The State Prediction Head takes $\mathbf{H}_t$ as its input, whereas the action tokens are decoded from the subsequent positions, which attend to the geometric tokens under causal attention. CAD-WAM thus realizes the joint prediction in Eq. (3) within a single model, in which both branches share the backbone and the Geometric State Embedding. Since gradients from both objectives propagate into these shared modules, action generation and state prediction jointly shape a common representation and thereby influence each other. Complete architectural details and hyperparameters are provided in Appendix C.

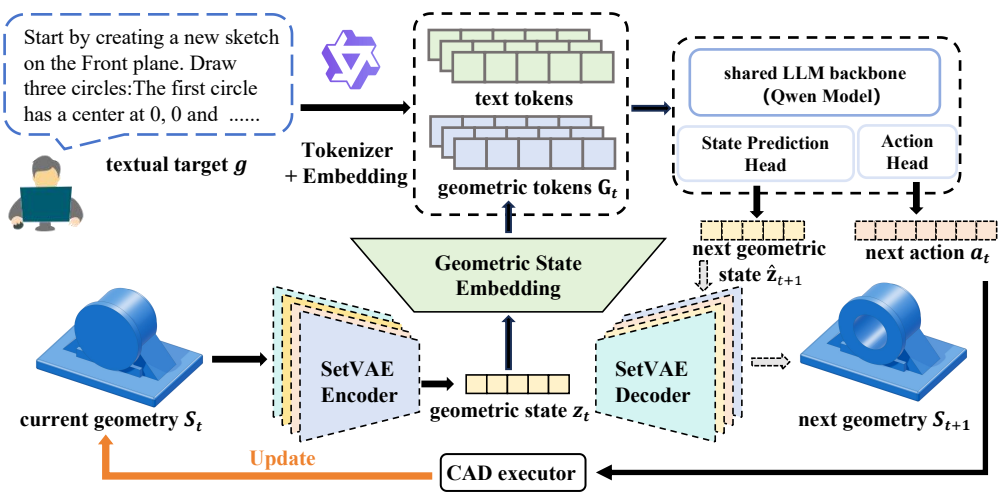


Figure 3: Overview of CAD-WAM, which builds the model step by step, executing each predicted action and re-encoding the resulting solid.

3.5 Training and Closed-Loop Inference

Training. We train CAD-WAM on expert trajectories, while the encoder E_ϕ obtained in Section 3.3 remains frozen, so that the regression target of state prediction stays fixed. Every step of a trajectory provides one supervised example: the geometric state $\mathbf{z}_t$ is obtained by encoding the solid that results from executing the first t ground-truth actions, and the supervision consists of the next action a_t and the next geometric state $\mathbf{z}_{t+1}$. Since all states are precomputed, the steps of a trajectory form independent examples and are processed in parallel. The two branches are trained jointly with

$$\mathcal{L} = -\sum_i \log \pi_\theta\big(a_t^{(i)} \mid a_t^{(<i)}, g, \mathbf{z}_t\big) + \frac{\lambda}{D}\big\|\hat{\mathbf{z}}_{t+1} - \mathbf{z}_{t+1}\big\|_2^2, \tag{6}$$

where $a_t^{(i)}$ is the i-th token of a_t and $D = 1024$ is the dimension of the geometric state. The objective combines two terms: a cross-entropy loss over the action tokens, which trains the Action Head, and a mean squared error between the predicted and the target geometric state, which trains the State Prediction Head. The weight λ balances the two terms.

Closed-Loop Inference. Figure 3 gives an overview of the complete pipeline. At inference, CAD-WAM constructs the CAD model step by step. At each step, the model receives the textual target g together with the geometric state $\mathbf{z}_t$ of the current model, and predicts the next action a_t and the next geometric state $\hat{\mathbf{z}}_{t+1}$. The predicted action is then executed by the CAD kernel, and the resulting solid is encoded by E_ϕ into $\mathbf{z}_{t+1}$, which the model receives at the next step. The loop is thus closed through the kernel and the encoder, so the state observed by the model always reflects the geometry that has actually been built. Generation terminates when the model emits the end action, when an action fails verification, or when a budget of $T_{\max}$ actions is reached, and the resulting solid is returned.

4 Step-Wise CAD Dataset

We construct a step-wise dataset from DeepCAD (Wu et al., 2021), a collection of 178,238 parametric CAD models, paired with the expert-level textual descriptions provided by Recap-OmniCAD (Qi et al., 2026), which serve as the textual targets g. We adopt the official DeepCAD split throughout.

Each DeepCAD program is converted into a sequence of actions in our DSL, and this sequence is executed step by step to obtain the intermediate models $S_0, \ldots, S_T$. Every intermediate model is sampled and encoded into its geometric state with the pretrained encoder E_ϕ (Section 3.3). Each step then yields one record: the textual target g and the current geometric state $\mathbf{z}_t$ form the input,

while the next action a_t and the next geometric state $\mathbf{z}_{t+1}$ provide the supervision. A final record supervising the end action completes each trajectory.

We retain a model only if it is annotated with a description and its executed geometry matches the reference model released with DeepCAD within a geometric tolerance. This yields 174,582 models and 526,010 records, of which 157,945 models with 475,656 records are used for training and 8,755 models with 26,394 records for validation. For evaluation, we use all 7,962 test models annotated with a description. Statistics of the resulting programs and an example record are provided in Appendix D.

5 Experiments

5.1 Experimental Setup

Evaluation Set. We evaluate on all 7,962 test models of the official DeepCAD split that are annotated with a textual description (Section 4), and compare each generated model against the reference solid released with DeepCAD.

Baselines. We compare CAD-WAM with two groups of methods. The first consists of task-specific text-to-CAD models: Text2CAD (Khan et al., 2024), CAD-Coder (Guan et al., 2025), CADFS (Pyatov et al., 2026), cadrille (Kolodiazhnyi et al., 2026), and Pointer-CAD (Qi et al., 2026). The second comprises general-purpose LLMs that generate CAD programs without task-specific training, namely Qwen3.8-Flash (Alibaba Cloud, 2026), DeepSeek-V4.1-Flash (DeepSeek-AI, 2026), GPT-6 Astra (OpenAI, 2026), and Claude Opus 5.5 (Anthropic, 2026). As these general-purpose LLMs are not trained for this task, we provide the DSL specification in the prompt and return three orthographic views of the solid built so far after each action. We generate the outputs of Text2CAD, CAD-Coder, and cadrille from their published checkpoints. For CADFS, we evaluate the inference results of the Qwen3-8B variant reported in its ablation study. For Pointer-CAD, we directly report its published results. Details of the baseline are provided in Appendix E.

Metrics and Protocol. Results reported in previous work are not directly comparable, since their evaluation protocols differ in the normalization of the solids (e.g., $[-1, 1]^3$ or $[-0.5, 0.5]^3$) and in the number of sampled points (from 8,192 to 100,000), among other aspects. We therefore re-score all systems under a unified protocol, in which every generated and reference solid is meshed, normalized, and sampled identically. Each task-specific baseline is evaluated with the textual annotations on which it was trained. All of them generate models of the same benchmark, since their training data derive from DeepCAD. The general-purpose LLMs are not trained for this task, and we therefore prompt them with the descriptions used in this work to generate DSL code, which is executed to obtain the corresponding solid. We adopt a comprehensive set of metrics covering both the validity and the geometric fidelity of the generated models, which are defined in detail in Appendix F.

Implementation Details. CAD-WAM is built on Qwen2.5-1.5B-Instruct (Qwen Team, 2024), whose parameters are fully fine-tuned together with the Geometric State Embedding and the State Prediction Head, with $\lambda = 0.03$. The encoder E_ϕ is pretrained as described in Section 3.3 and remains frozen. At inference, actions are decoded greedily, and generation is limited to $T_{\max} = 20$ actions. Training hyperparameters, hardware, and further details are reported in Appendix C.

5.2 Comparison with Existing Methods

Table 1 compares CAD-WAM with existing CAD generation methods and with general-purpose LLMs under the unified evaluation protocol. CAD-WAM attains the best value on every metric except the invalidity ratio, which is lowest for GPT-6 Astra. Among the task-specific methods it also generates valid models most reliably, with an invalidity ratio of 0.67% against 0.74% for the closest baseline. Its outputs agree with the references in structure rather than in shape alone, as reflected by the highest intersection over union and Euler characteristic match, and its mean Chamfer distance is 1.14 against 1.75 for the strongest general-purpose model.

As shown in Figure 4(a,b), the error distribution of CAD-WAM dominates those of the task-specific baselines over the entire range, and the difference is largest at the upper end, where the median

Table 1: Quantitative comparison with existing CAD generation methods and general-purpose large language models.

Metric		Text2CAD	CAD-Coder	CADFS-8B	Pointer-CAD	cadrille (text)	CAD-WAM (Ours)	Qwen3.8-Flash	DeepSeek-V4.1-Flash	GPT-6 Astra	Claude Opus 5.5
IR (%) ↓		2.627	0.741	8.125	8.800	2.529	0.666	28.360	29.478	**0.025**	9.068
CD ↓	*mean*	12.896	3.292	4.151	2.580	3.391	**1.142**	3.717	3.158	1.755	3.106
	median	0.554	0.210	0.196	0.300	0.218	**0.166**	0.206	0.200	0.173	0.197
HD↓	*mean*	14.446	6.576	7.491	–	7.365	**4.134**	6.634	6.304	4.728	6.354
	median	5.687	3.089	3.038	–	3.166	**2.783**	3.053	3.029	2.841	3.020
NC ↑	*mean*	0.693	0.845	0.839	–	0.836	**0.903**	0.854	0.860	0.888	0.859
	median	0.840	0.960	0.957	–	0.960	**0.964**	0.962	0.963	0.962	0.961
ECD ↓		45.301	12.289	11.606	–	14.128	**7.313**	13.137	12.124	8.186	11.112
IoU ↑		0.631	0.877	0.879	–	0.843	**0.964**	0.875	0.889	0.931	0.891
Euler match (%) ↑		80.712	91.596	85.960	–	86.706	**95.850**	92.089	92.840	95.434	91.879

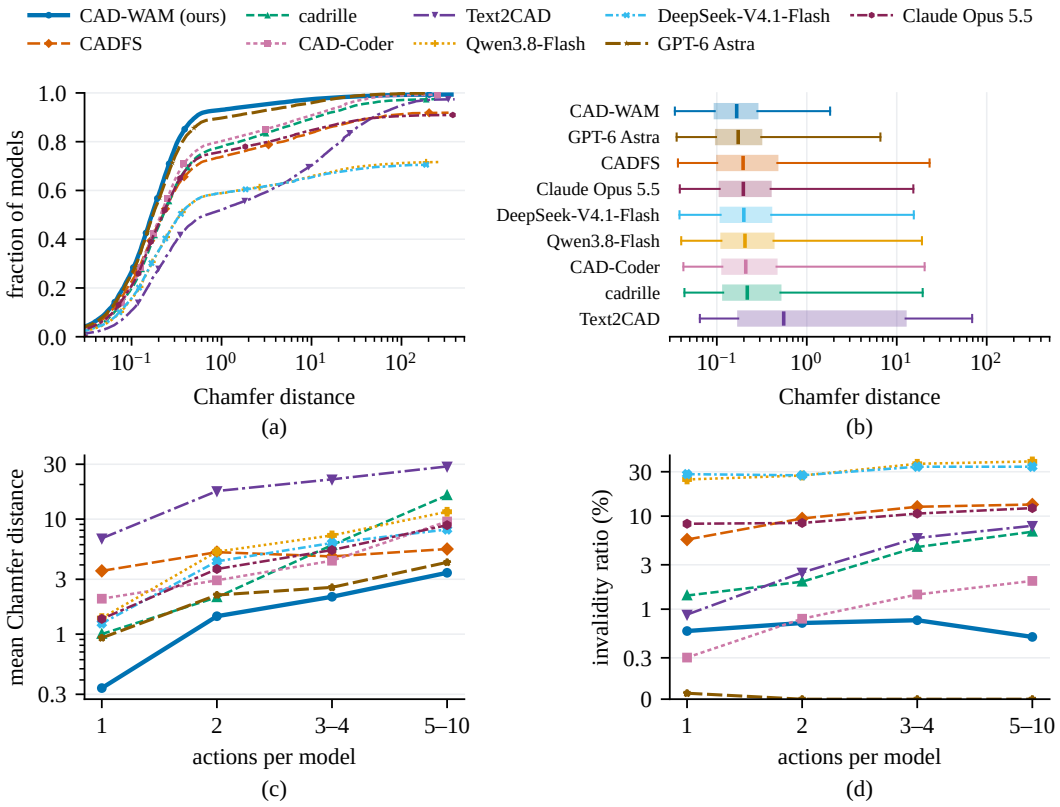


Figure 4: Per-model results for all compared systems. (a) Cumulative distribution of the Chamfer distance over the attempted models. (b) Quartile boxes with p5–p95 whiskers, ordered by median. (c) Mean Chamfer distance by number of actions. (d) Invalidity ratio by number of actions. Pointer-CAD is omitted, as its predictions are not accessible to us.

Chamfer distance decreases by 15% and the mean by 56% relative to the strongest baseline on each metric. Grouped by construction length (Figure 4(c,d)), CAD-WAM attains the lowest mean error in every group, and the gap to the baselines widens as models require more actions. The invalidity ratio of CAD-WAM remains low, whereas that of the baselines grows with the number of actions.

Table 2: Comparison between latent geometric states and textual states.

State	IR (%) ↓	CD ↓		HD ↓		NC ↑		ECD ↓	IoU ↑	Euler match (%) ↑
		mean	*median*	*mean*	*median*	*mean*	*median*			
Latent State	**0.666**	**1.142**	**0.166**	**4.134**	**2.783**	**0.903**	**0.964**	**7.313**	**0.964**	**95.850**
Text State	1.482	1.405	0.172	4.619	2.834	0.893	0.964	7.996	0.952	94.556

Table 3: Ablation study on the State Prediction Head.

Configuration	IR (%) ↓	CD ↓		HD ↓		NC ↑		ECD ↓	IoU ↑	Euler match (%) ↑
		mean	*median*	*mean*	*median*	*mean*	*median*			
w/o State Prediction Head	0.779	1.316	0.167	4.264	2.785	0.902	**0.964**	7.374	0.960	95.404
w/ State Prediction Head	**0.666**	**1.142**	**0.166**	**4.134**	**2.783**	**0.903**	**0.964**	**7.313**	**0.964**	**95.850**

5.3 Latent versus Textual Geometric States

To examine the role of the state representation, we train a variant in which the geometric state is expressed in text rather than encoded into a latent vector, and in which the next state is likewise generated as text (Appendix G). This variant therefore requires neither the Geometric State Embedding nor the State Prediction Head, and all remaining factors are identical.

As shown in Figure 4(a,b), the error distribution of CAD-WAM lies above those of the task-specific baselines over the entire range, and its quartile box and whiskers are the shortest of all systems; the difference is largest in the upper tail, where the median Chamfer distance decreases by 15% and the mean by 56% relative to the strongest baseline on each measure. Grouped by construction length (Figure 4(c,d)), CAD-WAM attains the lowest mean Chamfer distance in every group, and the gap widens as models require more actions. The invalidity ratio of CAD-WAM remains low across all groups, whereas that of the task-specific baselines grows with the number of actions.

5.4 Ablation on the State Prediction Head

To assess the contribution of state prediction, we train a variant in which the State Prediction Head is removed, so that the next geometric state is no longer predicted. All remaining settings are unchanged, so the two configurations differ only in the presence of this head.

As reported in Table 3, state prediction improves every aggregate metric. The invalidity ratio decreases from 0.78% to 0.67% and the mean Chamfer distance from 1.32 to 1.14, whereas the medians of the two configurations coincide. The improvement is therefore concentrated in the tail of the error distribution and is accompanied by a higher intersection over union and a higher Euler characteristic match. These results indicate that predicting the geometric state that should result from the next action, rather than generating the action alone, leads to more reliable generation.

6 Conclusion

We presented CAD-WAM, a text-to-CAD framework that couples geometry-conditioned action generation with next-state prediction. Its operation-level DSL and latent geometric states allow each modeling step to be learned through both an executable action and its intended geometric outcome. Next-state supervision shapes the shared representation used for action generation, while CAD-kernel execution provides feedback from the geometry actually built. On the DeepCAD test set, CAD-WAM achieves higher geometric accuracy than the evaluated task-specific and general-purpose models, with larger gains on models requiring more operations. Ablations show that the latent-state model outperforms its textual-state counterpart and that next-state supervision further improves generation. Together, these results support learning geometric evolution alongside command generation as an effective approach to text-to-CAD modeling.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish the writing of the paper, including grammatical correction and rephrasing for clarity and concision. We have reviewed all AI-assisted work: every assisted passage was read and, where necessary, rewritten by the authors, and all numbers and claims reported in the paper were checked against our own experimental records. We take responsibility for the final content of this work, including text, claims and artifacts.

REPRODUCIBILITY STATEMENT

The results of this paper rest on three components, each documented and released: the action representation together with the geometric executor that defines every state transition, the data construction and training procedure that learns from those states, and a single evaluation protocol applied uniformly to our model and to every baseline. The main paper states the model formulation and the experimental setting, while the Appendix specifies the metric definitions, the construction of the evaluation set, and the remaining implementation and ablation details. Since the same executor produces the states supervised during training and those fed back during closed-loop generation, our results can be reproduced end to end rather than only at the level of the final geometry. The supplementary material contains anonymous source code for the model, the data pipeline, trainingevaluation; public datasets and released baseline checkpoints are not redistributed, as our scripts rebuild every derived artifact from them.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikoł aj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a Visual Language Model for Few-Shot Learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 23716–23736. Curran Associates, Inc., 2022. doi: 10.52202/068431-1723.

Alibaba Cloud. Qwen3.8-Flash Released. Alibaba Cloud Model Studio product announcement, August 2026. URL https://www.alibabacloud.com/en/news/product/qwen38-flash-released-noi. Published 2026-08-28; accessed 2026-09-26.

Anthropic. Claude Opus 5.5 System Card. System card, 2026. URL https://www.anthropic.com/claude-opus-5-5-system-card. Accessed 2026-09-26.

Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Maria Elisabeth Bechtle, Feryal Behbahani, Stephanie C.Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando De Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative Interactive Environments. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4603–4623. PMLR, 2024.

CadQuery contributors. CadQuery. Computer software, version 2.7.0. Zenodo, 2026.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression. Technical report, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash. Accessed 2026-09-26.

Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning Universal Policies via Text-Guided Video Generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 9156–9172. Curran Associates, Inc., 2023. doi: 10.52202/075280-0403.

Prashant Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. CADmium: Fine-Tuning Code Language Models for Text-Driven Sequential CAD Design. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=lExqWvQht8.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-Coder: Text-to-CAD Generation with Chain-of-Thought and Geometric Reward. In *Advances in Neural Information Processing Systems*, volume 38, pp. 59765–59789. Curran Associates, Inc., 2025. doi: 10.52202/085713-1999.

David Ha and Jürgen Schmidhuber. World Models. *arXiv preprint arXiv:1803.10122*, 2018. doi: 10.48550/arXiv.1803.10122.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *International Conference on Learning Representations*, 2020.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating Sequential CAD Designs from Beginner-to-Expert Level Text Prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7552–7579. Curran Associates, Inc., 2024. doi: 10.52202/079017-0242.

Jinwoo Kim, Jaehoon Yoo, Juho Lee, and Seunghoon Hong. SetVAE: Learning Hierarchical Composition for Generative Modeling of Set-Structured Data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15059–15068, 2021.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal CAD Reconstruction with Reinforcement Learning. *arXiv preprint arXiv:2505.22914v3*, 2026. Version 3, 17 February 2026.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19730–19742. PMLR, 2023.

OpenAI. GPT-6 Astra System Card. System card, 2026. URL `https://deploymentsafety.openai.com/gpt-6-astra`. Accessed 2026-09-26.

Vladislav Pyatov, Gleb Bobrovskikh, Saveliy Galochkin, Nikita Boldyrev, Oleg Voynov, Alexander Filippov, Gonzalo Ferrer, Peter Wonka, and Evgeny Burnaev. CADFS: A Big CAD Program Dataset and Framework for Computer-Aided Design with Large Language Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10176–10186, 2026.

Dacheng Qi, Chenyu Wang, Jingwei Xu, Tianzhe Chu, Zibo Zhao, Wen Liu, Wenrui Ding, Yi Ma, and Shenghua Gao. Pointer-CAD: Unifying B-Rep and Command Sequences via Pointer-based Edges & Faces Selection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17377–17387, 2026.

Qwen Team. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024. doi: 10.48550/arXiv.2412.15115.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse Engineering CAD Code from Point Clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9801–9811, 2025. doi: 10.1109/ICCV51701.2025.00914.

Yang Tian, Sizhe Yang, Jia Zeng, Ping Wang, Dahua Lin, Hao Dong, and Jiangmiao Pang. Predictive Inverse Dynamics Models are Scalable Learners for Robotic Manipulation. In *International Conference on Learning Representations*, 2025.

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, 2021.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 24698–24724. PMLR, 2022.

Xiang Xu, Pradeep Kumar Jayaraman, Joseph George Lambourne, Karl D. D. Willis, and Yasutaka Furukawa. Hierarchical Neural Coding for Controllable CAD Model Generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38443–38461. PMLR, 2023.

Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Hao Li, Hengtao Li, Jie Li, Jindi Lv, Jingyu Liu, Min Cao, Peng Li, Qiuping Deng, Wenjun Mei, Xiaofeng Wang, Xinze Chen, Xinyu Zhou, Yang Wang, Yifan Chang, Yifan Li, Yukun Zhou, Yun Ye, Zhichao Liu, and Zheng Zhu. GigaWorld-Policy: An Efficient Action-Centered World–Action Model. *arXiv preprint arXiv:2603.17240*, 2026. doi: 10.48550/arXiv.2603.17240.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified World Models: Coupling Video and Action Diffusion for Pretraining on Large Robotic Datasets. In *Proceedings of Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.015.

APPENDIX

A CAD DOMAIN-SPECIFIC LANGUAGE

We detail here the action format introduced in Section 3.2 and the checks applied before and after execution.

A.1 ACTION SCHEMA

An action is a JSON object identified by its `kind` field; the extrusion used throughout our dataset is specified in Table 4, and Figure 1 shows such an action in its serialized form. Its sketch plane is given by an origin and two orthonormal directions, which define the local two-dimensional frame in which the profile is expressed. A profile consists of one or more closed loops, each a chain of curves in which every curve begins at the end point of its predecessor and the final curve closes the loop. Loops nested inside other loops define holes according to the even-odd rule. Lines are specified by their end point, arcs additionally by a sweep angle and an orientation flag, and circles by a center and a radius; a circle is always the only curve of its loop.

Table 4: Fields of an extrusion action.

Field	Type	Meaning
`kind`	string	operation identifier, `"extrude"`
`plane.origin`	3 floats	origin of the sketch plane
`plane.normal`	3 floats	plane normal, along which the profile is extruded
`plane.x_dir`	3 floats	in-plane x direction
`profile.loops`	list	closed loops of lines, arcs and circles
`forward`	float ≥ 0	extrusion distance along the normal
`backward`	float ≥ 0	extrusion distance against the normal
`op`	enum	`new`, `join`, `cut` or `intersect`
`taper`	float	draft angle in degrees, zero throughout DeepCAD

The pair (`forward`, `backward`) covers the three extrusion types of the source data: one of the two vanishes for a one-sided extrusion, both are equal for a symmetric one, and both are positive and distinct for a two-sided one. The Boolean operation then combines the extruded body with the current model. Because our executor fuses a new body with the existing model, `new` and `join` have the same geometric effect, and a `join` applied to an empty model is treated as `new`. A construction terminates with the dedicated action {`"kind": "end"`}.Figure 5 shows a complete action in its serialized form.

A.2 EXECUTION AND VALIDATION

Every action is validated before it reaches the CAD kernel. Beyond conformance to the schema, we require non-empty loops, a positive radius for a circle that forms its own loop, curves of non-zero length relative to the size of their loop, arc sweeps strictly between 0 and 2π, loops of non-zero area, a positive total extrusion distance, and a Boolean operation applicable to the current model, since `cut` and `intersect` are undefined on an empty model.

Actions that pass these checks are executed by the OpenCASCADE kernel through CadQuery (CadQuery contributors, 2026). A result is accepted only if it comprises at least one solid, passes the kernel's validity check, and has a volume exceeding 10^{-9} times the cube of its bounding-box diagonal. We additionally require the volume change to agree with the Boolean operation, within a relative tolerance of 10^{-6}: a `join` may not decrease the volume, and a `cut` or an `intersect` may not increase it. Violations trigger a single retry with a fuzzy Boolean tolerance of 10^{-6} times the bounding-box diagonal, after which the action is rejected if the result remains inconsistent. A rejected action leaves the current model unchanged, so that every accepted action yields a valid solid.

An action of our CAD DSL

```
{
  "kind": "extrude",
  "plane": {
    "origin": [0, 0, 0],
    "normal": [0, 0, 1],
    "x_dir": [1, 0, 0]
  },
  "profile": {
    "loops": [
      {"curves": [
        {"type": "line", "end": [0.0254, 0]},
        {"type": "arc", "end": [0, -0.0254],
         "sweep": 1.5708, "ccw": false},
        {"type": "line", "end": [0, -0.254]},
        {"type": "line", "end": [0.254, -0.254]},
        {"type": "line", "end": [0.254, 0]}
      ]}
    ]
  },
  "forward": 0.0127,
  "backward": 0,
  "op": "new",
  "taper": 0
}
```

Figure 5: An action of our DSL. A square profile with one rounded corner is drawn on the xy-plane and extruded as a new body.

B SetVAE Pretraining

The encoder E_ϕ of Section 3.3 is obtained by training SetVAE on the point clouds of the intermediate and final solids of the training split, that is 317,711 clouds from 157,945 models. A further 17,639 clouds of the validation split are used for checkpoint selection, and the clouds of the test split are never observed during this stage. Each cloud contains 2,048 points sampled over the surface of a solid, centered at its bounding box and scaled to a maximum half-extent of 0.75, after which the training-set mean and standard deviation are applied, following the original implementation.

The model follows the SetVAE architecture (Kim et al., 2021) with seven latent levels holding 1, 1, 2, 4, 8, 16 and 32 latent vectors of dimension 16, so that the concatenated posterior means form the 1024-dimensional geometric state. Training minimizes a Chamfer reconstruction term together with the Kullback-Leibler divergence, whose weight is annealed from zero to one over the first 40 epochs. We use Adam with a learning rate of 10^{-3}, a warm-up of 1,400 steps followed by a cosine decay to one percent of that value, a global batch of 256 clouds, and bf16 precision, for 300 epochs on eight H800 GPUs, which takes approximately 11.5 hours. Validation is performed every five epochs, and the checkpoint with the lowest validation Chamfer distance is retained; it occurs at the final epoch, with a validation Chamfer distance of 39.02 against 38.28 on the training clouds.

Figure 6 shows the training and validation curves. The transient increase of the validation error around the twentieth epoch coincides with the annealing of the Kullback-Leibler term, and the two curves remain close throughout, indicating that the model does not overfit the training clouds. Figure 7 compares input clouds of the test split with their reconstructions. Over 512 randomly selected test clouds, the reconstruction Chamfer distance is 2.44×10^{-3} on average, with a median of 1.96×10^{-3} and a 90th percentile of 4.74×10^{-3}, measured in the normalized frame in which each solid has a maximum half-extent of 0.75. Planar faces, holes and overall proportions are preserved, whereas fine edges are slightly rounded, which delimits the geometric detail that the state can represent. After this stage, E_ϕ is kept frozen and is used only to encode point clouds into geometric states.

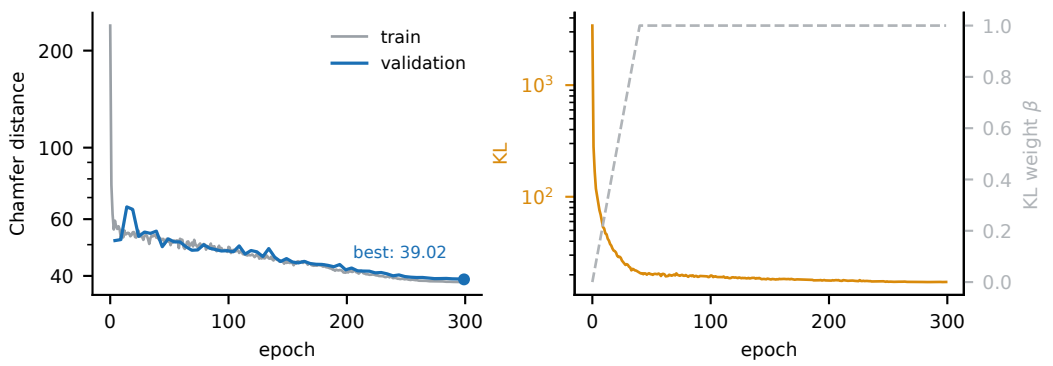


Figure 6: SetVAE pretraining. Left: training and validation Chamfer distance per epoch on a logarithmic scale, with the selected checkpoint marked. Right: the Kullback-Leibler term and its weight schedule.

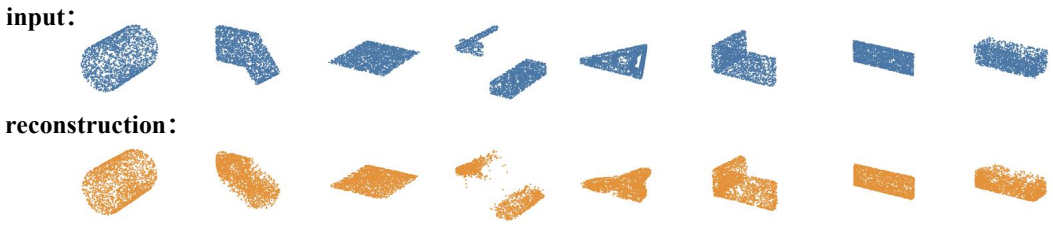


Figure 7: Reconstructions of test-split solids that the encoder never observed during pretraining, including intermediate solids from several construction steps. Input clouds are shown above their reconstructions.

C Implementation Details

Table 5 lists the configuration of CAD-WAM. All models are trained and evaluated on eight NVIDIA H800 GPUs with 80 GB of memory.

Fusion of the geometric tokens. The tokenized prompt contains sixteen contiguous reserved positions. Their token embeddings are replaced by the output of the Geometric State Embedding before the sequence enters the backbone. No layer of the decoder stack is modified. Gradients are not detached between the two branches, so both objectives update the backbone and the Geometric State Embedding.

Optimization. The cross-entropy is computed over the action tokens only, and the prompt is masked out. The state term is evaluated in single precision, independently of the compute precision. Fully sharded data parallelism wraps the model at the granularity of a decoder layer. All geometric states are precomputed and stored in a memory-mapped array, so each training step reads them without recomputation. The dataset is built so that no prompt exceeds the maximum sequence length. We validate on a fixed random subset of the validation split and report the cross-entropy of action tokens and state tokens separately.

Inference. Each action is decoded greedily and passed to a geometry worker. The workers run in a separate process and in the environment that produced the training point clouds, so the encoder observes the same tessellation at training and at test time. They are stateless, which isolates kernel failures from the generation process. The solid returned by a worker is sampled, normalized as in Appendix B, and encoded into the next geometric state. Test models are distributed over the eight GPUs, and the geometry workers of each shard run in parallel with generation.

Software environment. Training and generation use Python 3.12 with PyTorch 2.8.0 built against CUDA 12.8, Transformers 4.57.1 and FlashAttention 2.8.1, on GPUs with driver version 590.48.01.

Table 5: Configuration of CAD-WAM.

Architecture	
Backbone	Qwen2.5-1.5B-Instruct
Geometric tokens	16
Slot width	256
Geometric State Embedding	2 Transformer layers, 8 heads, feed-forward width 1024
State Prediction Head	mean pooling, LayerNorm, MLP $d \to 2048 \to 1024$ (GELU)
Added parameters	11.4M, or 0.74% of the backbone
Training	
Hardware	$8 \times$ NVIDIA H800
Precision and parallelism	bf16, fully sharded data parallelism
Epochs	1
Records per optimizer step	64, from micro-batch 4×8 GPUs $\times$ 2 accumulation
Optimizer steps	≈ 7.4k
Learning rate	2×10^{-5} for the backbone, 1×10^{-4} for the new modules
Schedule	3% warm-up, cosine decay to 10% of the peak
Gradient clipping	1.0
Maximum sequence length	5,632
State loss weight λ	0.03
Inference	
Decoding	greedy
Maximum tokens per action	2,048
Maximum number of actions	20
Points per solid	4,096 sampled, fixed subset of 2,048 encoded

Generation and evaluation are kept in separate environments: the geometry workers run the kernel version that produced the training data, whereas the evaluation environment applies one kernel version to the outputs of all systems.

D Dataset Construction

Source data. The CAD models and their construction sequences come from DeepCAD (Wu et al., 2021), which provides 178,238 models together with a reference solid for each of them. The textual targets are the expert-level descriptions provided by Recap-OmniCAD (Qi et al., 2026). We keep DeepCAD's official split and do not remove duplicate programs.

Conversion to the DSL. Every sketch profile and its extrusion are converted into one action of our DSL. The sketch plane is taken from the transform of the sketch, and the loops of the profile are taken from its curves. The two extrusion distances follow the extent type of the feature, which is one-sided, symmetric or two-sided. The Boolean operation follows the feature operation of the original model. An extrusion that references several profiles is split into a sequence of single-profile actions, which leaves the resulting geometry unchanged for the operations that occur in DeepCAD. All coordinates are kept in the units of the source data, and no rescaling is applied.

Verification. Each converted program is executed and the resulting solid is compared with the reference solid. A model is retained when the maximum surface distance between the two solids is below 10^{-6}. This procedure verifies 176,226 of the 178,238 models, and 174,582 of those also carry a description. The remaining models fail because the conversion does not complete, because the executor rejects an action, or because the resulting geometry differs from the reference. Table 6 summarizes the construction of the dataset.

Step-wise records. Each program is executed one action at a time, which yields the intermediate solids of the construction. Every intermediate solid is sampled and encoded into a geometric state with the pretrained encoder of Appendix B. A record is then formed for every step, and a final record supervises the end action. Of the 526,010 records, 351,428 correspond to a modeling action and the remainder to the end action. The records are stored in columnar files, and the geometric states are stored in a memory-mapped array whose first row holds the zero state of the empty model.

Table 6: Construction of the step-wise dataset. The test split lists the models for which step-wise records exist, whereas the evaluation set comprises all test models that carry a description, including 80 models that do not pass verification.

	Models	Records
DeepCAD models	178,238	
Geometrically verified	176,226	
With an expert description	174,582	526,010
Training split	157,945	475,656
Validation split	8,755	26,394
Test split	7,882	23,960
Evaluation set	7,962	

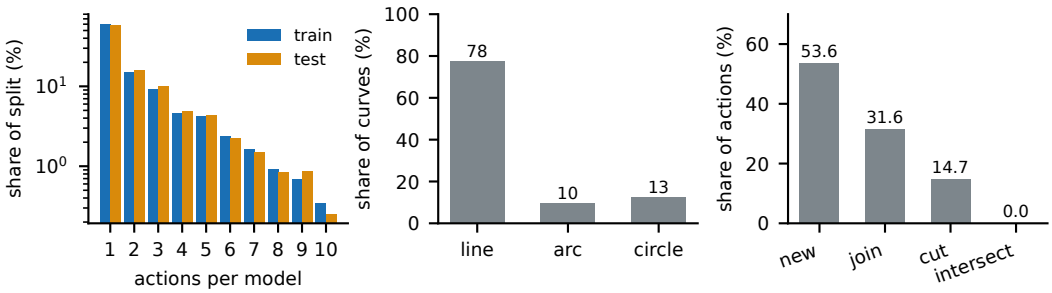


Figure 8: Statistics of the step-wise dataset. Left: number of actions per model, shown as a share of the training and of the test split on a logarithmic scale. Middle: composition of the sketch profiles by curve type. Right: distribution of the Boolean operations over all actions.

A record can only be built for a model whose program executes and matches its reference, which is why the test split of the dataset is limited to the verified models. Generation, in contrast, starts from the empty model and requires only the description. We therefore evaluate on every test model that carries a description, so that the evaluation is not restricted to the models our conversion reproduces.

Statistics. Figure 8 summarizes the resulting programs. The number of actions per model ranges from one to ten, and the test split follows the same distribution as the training split. Sketch profiles consist of 1,139,814 lines, 184,224 circles and 142,333 arcs, and 279,304 profiles contain a single loop. Among the Boolean operations, 188,528 create a new body, 111,139 join, 51,651 cut and 110 intersect. Every action in DeepCAD uses a zero draft angle. Figure 9 shows a record of the dataset.

E Baseline Configurations

We generate the outputs of Text2CAD, CAD-Coder and cadrille ourselves from the checkpoints released by their authors, and we use the released inference results of CADFS. For Pointer-CAD we report published numbers, since its generated models are not accessible to us. The general-purpose models are queried through their APIs with the step-wise protocol described below. Table 7 summarizes the configurations.

Text2CAD. We use the official checkpoint and the expert-level prompts released with the model, and decode greedily. Prompts are unavailable for 121 test models, which are therefore not attempted. Sequences are converted to solids with the post-processing used in the authors' evaluation script. Under our protocol this configuration reproduces the metrics reported in the original paper, which confirms that the pipeline is faithful.

CAD-Coder. We use the official checkpoint and the released test file, with the sampling settings of the authors: temperature 0.7, nucleus sampling with $p = 0.7$, a limit of 4,096 tokens, and a fixed seed. No generation is truncated, and 99.5% of the generated programs execute.

Input

```
You are constructing a CAD model step by step with cadseq version 1 actions.  You are given
the CAD description of the target model and the current geometric state of the partially
built model, encoded as 16 geometric state tokens.  Generate ONLY the next cadseq action
as a single JSON object, preserving the described dimensions.  If the model is already
complete, return {"kind":  "end"}.

CAD description:
To construct the CAD model of the bracket, follow these detailed steps:
### Part 1:  Symmetrical Square Bracket with Rounded Edges and Central Cutout
1.  Create Sketch 1 on the Top Plane:  draw an outer loop of four arcs and four lines [...]
and an inner loop consisting of a single circle of radius 0.0025.
2.  Extrude the sketch to create a new body, extending 0.003 in the positive normal
direction.
### Part 2:  Central Circular Cutout with Smaller Concentric Ring [...]

Actions completed so far:  1
Current bounding box:  size 0.02 x 0.02 x 0.003, center (0, 0, 0.0015)
Current geometric state:
<GEO>[16 geometric tokens]</GEO>
```

Target action

```
{"kind":  "extrude",
  "plane":  {"origin":  [0, 0, 0.003], "normal":  [0, 0, 1],
      "x_dir":  [1, 0, 0]},
  "profile":  {"loops":  [
    {"curves":  [{"type":  "circle", "center":  [0, 0], "radius":  0.0025}]},
    {"curves":  [{"type":  "circle", "center":  [0, 0], "radius":  0.005}]}]},
  "forward":  0, "backward":  0.0015, "op":  "cut", "taper":  0}
```

Figure 9: A record of the step-wise dataset, taken from the test split. The input contains the instruction, the description of the target model, the progress information and the geometric state of the current solid; the description is shortened here. The target is the next action, which cuts a ring into the bracket. The supervision also includes the geometric state of the solid that this action produces.

Table 7: Configuration of the compared systems. "Attempted" is the number of test models the system was run on.

System	Outputs	Generation	Attempted
Text2CAD	our inference	single pass	7,841
CAD-Coder	our inference	single pass	7,958
cadrille	our inference	single pass	7,949
CADFS	released results	single pass	7,212
Pointer-CAD	published results	step-wise	–
Qwen3.8-Flash	API	DSL, rendered views	7,962
DeepSeek-V4.1-Flash	API	DSL, rendered views	7,962
GPT-6 Astra	API	DSL, rendered views	7,962
Claude Opus 5.5	API	DSL, rendered views	7,962
CAD-WAM	our model	DSL, latent state	7,962

cadrille. We use the released checkpoint in its text-conditioned mode with the decoding settings of the authors.

CADFS. We evaluate the released inference results of the Qwen3-8B variant reported in the ablation study of the original paper. Of the attempted models, 586 produce no solid and are counted as invalid.

Pointer-CAD. The published results are obtained on the same test models and with the same Chamfer distance convention as ours, which makes them comparable to the re-scored systems. Metrics that the paper does not report are left blank in Table 1.

General-purpose language models. These models are not trained for this task, so we provide the specification of our DSL and let them follow the same step-wise protocol as CAD-WAM. Each test model is produced in one conversation. The first turn contains the grammar of the action language, including the plane conventions of the source data, together with the description of the target model, and the model replies with one action per turn. After each reply the kernel executes the actions produced so far. If an action fails to execute, generation stops for that model, as in our own inference loop. Otherwise the resulting solid is rendered as three orthographic views, from the front, the top and the right, and returned with its measured bounding box; this rendering takes the place of the latent geometric state that CAD-WAM observes. A conversation ends when the model emits the end action, when an action fails, or after at most 12 actions, and the solid built so far is exported. A model is counted as invalid when no solid can be exported. Decoding uses temperature zero, and the views are rendered at 512 pixels.

F Evaluation Protocol

Published results are not comparable across papers, because each defines its metrics on a different normalization and a different number of samples. We therefore evaluate every system with a unified protocol, described below. Each metric follows the definition of the paper that introduced it, so that our values remain on the established scale.

Meshing and sampling. Let P denote a predicted solid and R the corresponding reference. Both are fitted into $[-1, 1]^3$ by centering their bounding box and scaling by $2/\ell$, where ℓ is the longest side of that box. This fitting makes the meshing tolerances and the Boolean volumes comparable across models of different physical size. The solids are then tessellated with a linear deflection of 0.001 and an angular deflection of 0.5. From each mesh we draw $N = 8{,}192$ samples uniformly with respect to area, which gives the point sets $X = \{\mathbf{x}_i\}$ and $Y = \{\mathbf{y}_j\}$ together with the normals of the faces they were sampled from. These sets are used for the Chamfer distance, the Hausdorff distance and the normal consistency.

Normalization of the point sets. Each metric applies the normalization of its original definition, which we keep so that our values can be read on the published scale. For the Chamfer and Hausdorff distances, each cloud is centered on its mean and divided by the range of its coordinates, taken over all three axes jointly, which places it in approximately $[-0.5, 0.5]^3$. For the edge Chamfer distance, each cloud is centered on its mean and divided by the largest absolute coordinate, which places it in approximately $[-1, 1]^3$. The two conventions differ by a factor of about two in linear distance, hence by about four in squared distance, but each is applied identically to every system, so the comparison between systems is unaffected.

Chamfer distance: overall surface agreement. The Chamfer distance measures how closely the two surfaces agree on average. With $d(\mathbf{x}, Y) = \min_{\mathbf{y}\in Y} \|\mathbf{x} - \mathbf{y}\|$,

$$\mathrm{CD}(X, Y) = 10^3 \left(\frac{1}{|X|} \sum_{\mathbf{x}\in X} d(\mathbf{x}, Y)^2 + \frac{1}{|Y|} \sum_{\mathbf{y}\in Y} d(\mathbf{y}, X)^2 \right). \tag{7}$$

Both directions are summed and the result is scaled by 10^3, following the convention of the compared work. Squaring the distances makes the measure sensitive to large local deviations, which is why we report the median as well as the mean.

Hausdorff distance: worst-case deviation. The Hausdorff distance reports the largest deviation between the two surfaces rather than the average,

$$\mathrm{HD}(X, Y) = 10^2 \max\Big\{ \max_{\mathbf{x}\in X} d(\mathbf{x}, Y),\ \max_{\mathbf{y}\in Y} d(\mathbf{y}, X) \Big\}. \tag{8}$$

It is evaluated on the same point sets as the Chamfer distance and is not squared; the factor 10^2 is applied to every evaluated system.

Normal consistency: orientation of the surfaces. Normal consistency measures whether the surfaces agree in orientation and not only in position, which distinguishes a correctly reconstructed face from a nearby but differently oriented one. It is computed on the same $8{,}192$-point sets. Let $\mathbf{n}_{\mathbf{x}}$ be the normal of the face from which $\mathbf{x}$ was sampled and let $\pi_Y(\mathbf{x})$ denote the nearest neighbor of $\mathbf{x}$ in Y. Then

$$\mathrm{NC}(X,Y) = \frac{1}{2}\left(\frac{1}{|X|}\sum_{\mathbf{x}\in X}\left\langle \mathbf{n}_{\mathbf{x}}, \mathbf{n}_{\pi_Y(\mathbf{x})}\right\rangle + \frac{1}{|Y|}\sum_{\mathbf{y}\in Y}\left\langle \mathbf{n}_{\mathbf{y}}, \mathbf{n}_{\pi_X(\mathbf{y})}\right\rangle\right). \tag{9}$$

Edge Chamfer distance: fidelity of sharp features. Mechanical parts are characterized by their edges, which a surface metric weights only weakly. This metric therefore compares the sharp edges alone, under its own sampling. Both shapes are sampled with 10^5 points and normalized as described above. A point is assigned to an edge when a neighbor within a radius of 0.004 has a normal satisfying $|\langle \mathbf{n}, \mathbf{n}'\rangle| < 0.2$. The edge Chamfer distance is Eq. (7) evaluated on the two sets of edge points, and we report its median.

Intersection over union: agreement of the occupied volume. This metric measures how much of the material of the two solids coincides, which a surface metric does not capture. Both solids are fitted into $[-1,1]^3$, and their intersection and union are obtained with Boolean operations of the kernel, evaluated with a fuzzy tolerance, while the volumes V_P, V_R, $V_\cap$ and $V_\cup$ are the exact volumes of the resulting solids,

$$\mathrm{IoU}(P,R) = \frac{V_\cap}{V_\cup}. \tag{10}$$

Boolean operations on nearly coincident faces are numerically delicate, so each result must satisfy the inclusion-exclusion identity

$$\left|\, V_\cup - (V_P + V_R - V_\cap)\,\right| \le 10^{-3}\max\{V_\cup, V_P, V_R\}. \tag{11}$$

We evaluate the Booleans with a fuzzy tolerance of 10^{-5}, then without a tolerance, then with 10^{-4}, and keep the first result that satisfies Eq. (11). Pairs for which no setting is consistent are excluded and their number is reported.

Euler characteristic match: topological agreement. This criterion checks whether the generated model has the same topology as the reference, for instance the same number of holes. It is evaluated only when both meshes are watertight, and equals one when their Euler characteristics agree after unreferenced vertices have been removed.

Invalidity ratio: whether a usable model is produced at all. A prediction is counted as invalid when its solid cannot be loaded, has zero volume, or cannot be meshed. The invalidity ratio is the share of such predictions among the models a system attempted,

$$\mathrm{IR} = 1 - \frac{|\{\text{valid predictions}\}|}{|\{\text{attempted models}\}|}. \tag{12}$$

A model counts as attempted when the system was run on it and, for the systems whose outputs we obtained from their authors, when an output was released for it. CAD-WAM and the zero-shot models attempt all 7,962 evaluation models, CAD-Coder attempts 7,958, cadrille 7,949, Text2CAD 7,841 and CADFS 7,212, as listed in Appendix E. All other metrics are computed on the valid predictions whose reference can be scored. They are therefore conditional on a system producing a solid at all, and must be read together with the invalidity ratio, which is defined over the same set of attempted models.

G Textual State Variant

This section specifies the variant of Section 5.3, in which the geometric state is expressed in text instead of being encoded into a latent vector.

Architecture. The variant uses the backbone without modification. Since the state is part of the text, neither the Geometric State Embedding nor the State Prediction Head is required, and the model has no additional module and no continuous input. It reads a prompt and produces text, exactly as a language model trained for any other generation task. The target combines the next action and the next state in a single JSON object, so the variant still predicts the state that its action produces, and it is trained with a cross-entropy loss over both. The weight λ therefore does not apply.

Content of the textual state. The textual state is a deterministic summary of the current solid, expressed in the coordinate frame and the units of the actions. It reports the number of solids, the bounding box, the volume, the surface area, and the numbers of faces, edges and vertices. It further lists the faces in decreasing order of area. Each entry gives the type of the face and the parameters of that type: a plane is described by its center and its normal, and a curved face by its center, its radius and its convexity, together with the axis of a cylinder, a cone or a torus, the half-angle of a cone, and the minor radius of a torus. Numbers are written with five significant digits, and the faces are ordered after rounding, so that a given geometry always yields the same text. The empty model is represented by `{"solids": 0}`. Figure 10 illustrates the state of a solid with seven faces.

```
{"solids": 1,
 "bbox": {"min": [-0.035139, 0.0, -0.023724],
          "max": [0.060353, 0.095213, 0.025922]},
 "volume": 0.00033102, "area": 0.046592,
 "faces": 7, "edges": 15, "vertices": 10,
 "face_list": [
   {"type": "cylinder",
    "center": [-0.0090427, 0.047607, 0.0],
    "axis": [0, 1, 0], "radius": 0.02006,
    "convex": false, "area": 0.012001},
   {"type": "plane",
    "center": [0.012607, 0.047607, -0.023724],
    "normal": [0, 0, -1], "area": 0.0090921},
   {"type": "plane",
    "center": [0.012607, 0.047607, 0.025922],
    "normal": [0, 0, 1], "area": 0.0090921},
   ...
 ]}
```

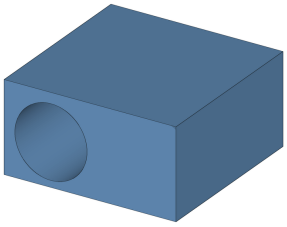


Figure 10: Textual state of an intermediate solid and the solid it describes. The part is a block with a cylindrical through-hole, which the state reports as six planar faces and one concave cylindrical face; three of the seven faces are shown.

Input and target. The prompt contains the same instruction, the same description and the same progress information as in the main model, and the textual state replaces the geometric tokens. The target is a JSON object holding the next action and the state that this action produces. At the end of a construction, the action is the end action and the state is left unchanged.

Training and cost. The variant is trained on the same records, with the same backbone and the same optimization settings as the main model. The textual state dominates the input and the target: it accounts for about four fifths of the tokens of a target, and the length of a record grows with the number of faces of the current solid.

H Effect of the State Prediction Weight

The weight λ of Eq. (6) sets the relative contribution of the auxiliary state objective. We train one model per value and evaluate all of them with the protocol of Appendix F; every other setting is identical.

As reported in Table 8, every positive weight improves upon $\lambda = 0$ across the metrics that characterize geometric quality, namely the Chamfer and Hausdorff distances, the normal consistency and the edge Chamfer distance, and the intersection over union improves as well. The remaining metrics vary within a narrow band around their values at $\lambda = 0$. The differences among the positive weights

Table 8: Ablation study on the state prediction loss weight λ. CD, HD, and NC are reported as mean values.

Metric	$\lambda = 0$	$\lambda = 0.01$	$\lambda = 0.03$	$\lambda = 0.05$	$\lambda = 0.1$
IR (%) ↓	0.779	0.691	**0.666**	0.791	0.804
CD ↓	1.316	1.229	1.142	1.232	**1.107**
HD ↓	4.264	4.203	**4.134**	4.163	4.184
NC ↑	0.902	0.902	**0.903**	0.903	0.903
ECD ↓	7.374	7.251	7.313	**7.189**	7.196
IoU ↑	0.960	0.961	**0.964**	0.963	0.962
Euler match (%) ↑	95.404	95.333	**95.850**	95.797	95.629

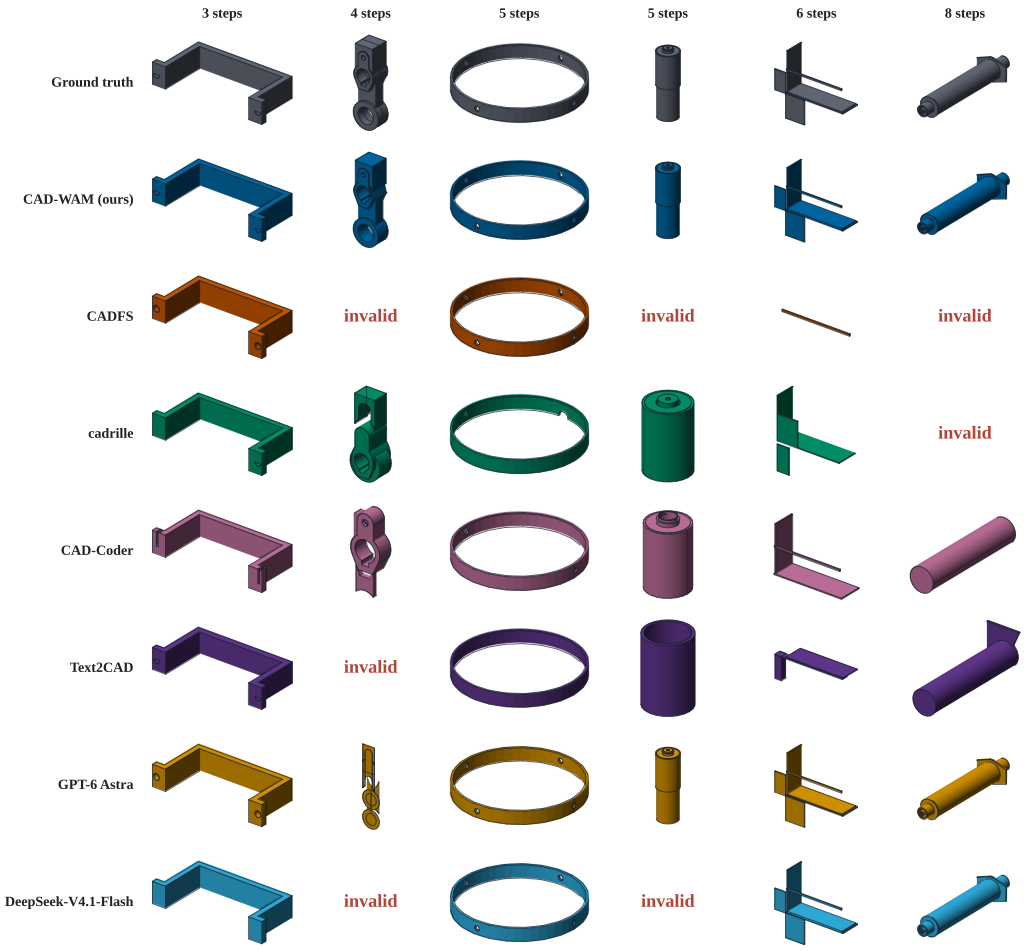


Figure 11: Predictions of each system on six test models, ordered by the number of construction steps of the reference. *invalid* marks a prediction that does not build a valid solid.

are small, so state prediction acts as an auxiliary signal whose benefit does not depend on a precise weight. We adopt $\lambda = 0.03$ in all other experiments.

I Qualitative Examples

Figure 11 compares the solids produced by every system on six models of the evaluation set whose references require three to eight construction steps. Each row corresponds to one system and each

column to one model, with the reference in the first row. Every solid is fitted to $[-1, 1]^3$, the normalization under which the metrics of Appendix F are computed, and all cells of a column are rendered from a single camera, so that the predictions of different systems can be compared directly.

On the model with the fewest steps every system recovers the intended shape, which is the regime that the median over the full evaluation set reflects. As the number of steps increases, the predictions of the baselines depart from the reference in two ways: they collapse to a single primitive that matches the overall proportions but none of the detail, such as a plain cylinder in place of a stepped shaft or a flat plate in place of a finned bracket, or they fail to produce a valid solid at all. CAD-WAM builds a valid solid for all six models and reproduces the features introduced by the later steps, with a Chamfer distance below 0.15 throughout. These examples illustrate at the level of individual models the behavior observed in aggregate on the full evaluation set: the advantage of predicting the geometric state grows with the number of operations a model requires.

J LIMITATIONS

Our evaluation is limited to the sketch-and-extrude programs in DeepCAD. Extending CAD-WAM to richer operations and longer construction sequences would help assess its applicability to more demanding modeling tasks. The state prediction head currently supplies supervision only. Since it estimates the outcome of an action before execution, it can equally serve as a look-ahead signal: candidate actions are scored against their predicted states, and the model commits to one only after that comparison.