
ForgeCAD: Reformulating Drawings-to-CAD Generation as a Symbolic Routing Task

Shurui Liu¹ Yihao Guo² Weide Chen³ Donghao Wu¹ Ancong Wu^{1*}

¹ School of Computer Science and Engineering, Sun Yat-sen University, China ² Hunan University

³ School of Intelligent Systems Engineering, Shenzhen Campus of Sun Yat-sen University, China

{liushr29, chenwd56, wudh53}@mail2.sysu.edu.cn

guoyihao@hnu.edu.cn wuanc@mail.sysu.edu.cn

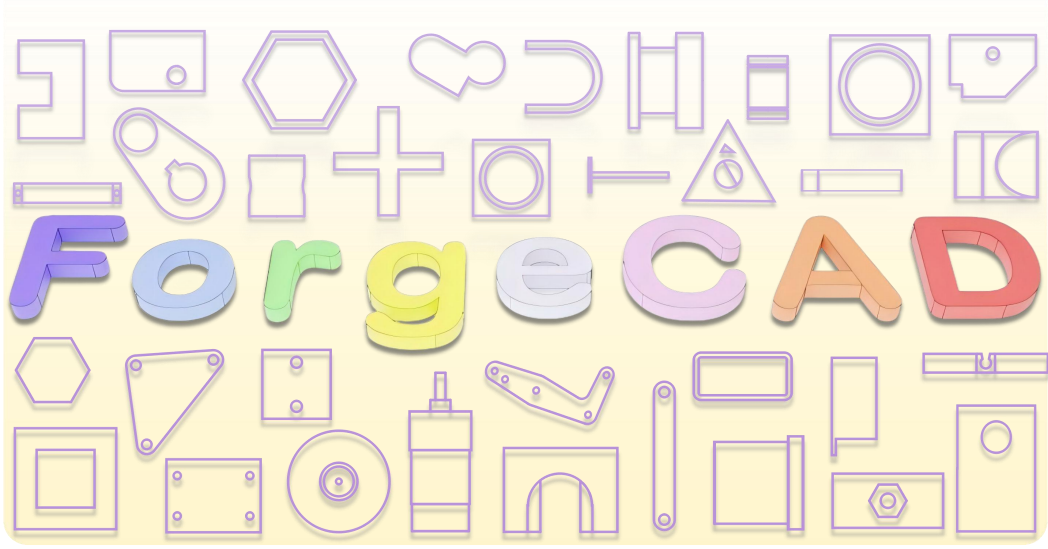


Figure 1: Bridging 2D and 3D in industrial design. Visualizing the complexity of the Drawings-to-CAD generation task. The examples illustrate the strict geometric precision required to translate 2D orthographic projections into procedurally valid 3D CAD models.

Abstract

Computer-Aided Design (CAD) generative modeling is indispensable for modern industrial manufacturing, where 2D orthographic engineering drawings traditionally serve as the foundational specifications for precision design. While recent deep learning advancements have accelerated 3D asset generation, existing Drawings-to-CAD frameworks fundamentally formulate this translation as a numerical regression problem. By relying on synthesizing geometric parameters from scratch, these methods struggle to capture strict topological constraints, frequently producing invalid geometries and structural artifacts when handling dense components. To overcome these limitations, we present **ForgeCAD**, a novel framework that reformulates Drawings-to-CAD generation as a symbolic routing task. Specifically, we introduce a pointer-centric symbolic representation that extracts all geometric properties from the engineering drawings and instantiates them exactly once into a unified geometric table. Under this formulation, we demonstrate that standard Large Language Models (LLMs) can inherently function as powerful

* Corresponding author.

pointer networks without specialized network engineering. By decoupling CAD generation to optimize for logical symbol retrieval, our model synthesizes CAD models by purely referencing and composing existing pointers from the shared dictionary. Extensive experiments show that **ForgeCAD** achieves state-of-the-art performance, delivering superior geometric fidelity, structural correctness, and seamless alignment with original design intents.

1 Introduction

Computer-Aided Design (CAD) modeling is the backbone of modern industrial manufacturing [66, 39]. Unlike general 3D asset generation [23, 57, 36, 17, 71, 54, 38, 37], engineering design demands rigorous geometric precision and strict topological constraints. In standard industrial workflows, orthographic engineering drawings serve as the definitive starting point [4, 9, 46], conveying exact dimensional specifications and spatial relationships. This practice naturally motivates a critical but highly complex task [52, 15]: automating the translation of 2D orthographic engineering drawings into procedural 3D parametric CAD models.

Early mathematical frameworks and geometric reasoning approaches established important theoretical foundations for this 2D-to-3D translation [11, 22, 32, 51, 10], though they were typically restricted to simple geometries. Subsequent rule-based and learning-based methods advanced the field by recognizing geometric features during the modeling process [4, 13]. Nevertheless, due to their reliance on rasterized images, these methods inherently suffer from a loss of geometric precision and fidelity. To address these issues, recent work has pushed the boundary by utilizing orthographic engineering drawings in Scalable Vector Graphics (SVG) format as the input modality for CAD generation [41].

Despite their strong empirical performance, none of these existing methods formulate the translation from orthographic drawings to 3D CAD models as a symbolic routing task. Instead, existing Drawings-to-CAD frameworks fundamentally treat the translation process as a numerical regression problem [41]. Driven by complex, task-specialized architectures, these approaches attempt to synthesize geometric parameters and spatial coordinates entirely from scratch. These approaches fail to enforce natural, rigorous topological and geometric consistency between the 2D engineering drawings and the generated 3D CAD models. When handling complex industrial designs characterized by dense and overlapping primitives, existing numerical approaches frequently suffer from severe spatial hallucinations, conflated vertices, and open boundary representations.

To overcome this fundamental limitation, we present **ForgeCAD**, a novel framework that reformulates 2D-to-3D CAD generation from a regression problem into a symbolic routing task. Instead of predicting raw numerical values, we introduce a pointer-centric symbolic representation. We extract all geometric parameters, including spatial coordinates, lengths, and radii, from the vector drawings and instantiate them exactly once into a unified reference table. Consequently, the model generates the 3D CAD program by simply selecting, reordering, and composing existing geometric pointers from this shared dictionary.

ForgeCAD bypasses brittle, hand-crafted architectural modifications. We demonstrate that standard Large Language Models (LLMs) can inherently function as powerful pointer networks under our representation. By decoupling the generation objective to explicitly optimize for logical symbol retrieval, **ForgeCAD** leverages the capacity of general-purpose sequence modeling, naturally resolving the spatial ambiguities that plague prior methods.

Overall, our contributions are:

- We reformulate Drawings-to-CAD generation as a symbolic routing task. By introducing a novel pointer-centric representation, we unify geometric parameters into a shared reference table, effectively mitigating spatial hallucinations and geometric ambiguities.
- We propose **ForgeCAD**, a framework that fine-tunes LLMs to function as pointer networks without requiring specialized architectural modifications, effectively leveraging the capacity of scalable sequence modeling for rigorous topological and geometric consistency.

- Extensive experiments demonstrate that **ForgeCAD** achieves state-of-the-art performance. Our method exhibits remarkable robustness, high-fidelity, and precise alignment with the definitive design intents of orthographic engineering drawings.

2 Related Work

2.1 CAD Generation and Reconstruction

Driven by deep learning, CAD modeling has seen significant progress, advancing from the introduction of foundational datasets [55, 20, 53] to recent generative and reconstructive pipelines [29, 59, 27, 24, 28, 56, 3, 55, 60, 25, 31, 18, 42, 5, 30, 48, 3, 63, 19]. Current methods predominantly focus on reverse engineering, reconstructing CAD models from point clouds [55, 60, 25, 31, 18, 42] or single- and multi-view images [5, 30]. Concurrently, forward design approaches driven by text [48, 3, 63, 19] or multimodal inputs [58, 33] have also pioneered new generation capabilities. However, these paradigms often struggle to capture the strict geometric precision required in standard industrial workflows. In practice, orthographic engineering drawings serve as the definitive starting point for CAD design, naturally motivating the task of translating 2D engineering drawings into 3D CAD models.

Early mathematical frameworks and geometric reasoning approaches established important theoretical foundations for this 2D-to-3D translation, though they were typically restricted to simple geometries [11, 22, 32, 51, 10]. Subsequent rule-based [4, 13] and learning-based methods [40, 52, 65] advanced the field by recognizing geometric features during the modeling process. Nevertheless, due to their reliance on rasterized images, these methods inherently suffer from a loss of geometric precision and fidelity.

To address these issues, recent work has pushed the boundary by utilizing orthographic engineering drawings in SVG format as the input modality for CAD generation [41]. Despite their strong empirical performance, none of these existing methods formulate the translation from orthographic drawings to 3D CAD models as a symbolic routing task. By treating generation as numerical prediction rather than discrete symbolic routing, previous approaches fail to enforce natural, rigorous topological and geometric consistency between the 2D engineering drawings and the generated 3D CAD models.

2.2 Large Language Models (LLMs)

Compared to traditional deep learning architectures, Large Language Models (LLMs) have demonstrated an exceptional capacity for instruction tuning and in-context learning [45, 2, 62, 6, 47]. This has unlocked notable versatility across diverse domains [26, 68, 43, 67, 26, 16], driving state-of-the-art performance in complex reasoning tasks such as code generation [12, 7] and question answering [44, 21]. More recently, both LLMs and Vision-Language Models (VLMs) have been adapted for CAD generation [58, 3, 64, 49, 50, 70, 69]. These approaches typically rely on captioning 3D models or fine-tuning language models to generate CAD command sequences and code from human descriptions. However, natural language is inherently ambiguous; consequently, these text-driven methods frequently fail to enforce the rigorous topological and geometric consistency demanded by real-world design intents. **ForgeCAD** fundamentally departs from this paradigm. By abandoning imprecise linguistic descriptions in favor of orthographic engineering drawings as the primary input modality, our approach largely bypasses spatial heuristics. This promotes high-fidelity geometric generation while naturally aligning with established industrial design workflows, which universally rely on precision orthographic projections as their definitive starting point.

3 Methodology

In this section, we present **ForgeCAD**, our proposed framework for translating 2D orthographic drawings into procedural 3D CAD models. We fundamentally reframe the CAD reconstruction pipeline as a fully discrete symbolic routing task. By abandoning numerical regression, we allow the model to focus purely on structural topology and procedural logic. The remainder of this section is structured as follows: Sec. 3.1 formally introduces our pointer-centric symbolic representation, which extracts and unifies all geometric parameters into a shared reference table. Following this, Sec. 3.2

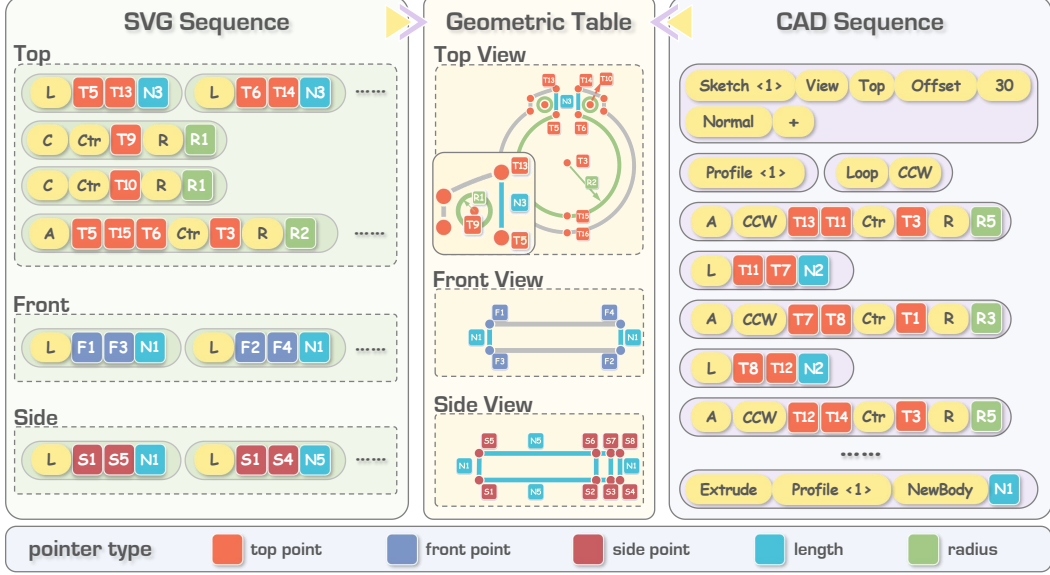


Figure 2: **Overview of the pointer-centric symbolic representation.** We bridge the gap between orthographic SVGs and CAD sequences by mapping both into a shared table composed by geometric primitives (points, lengths, and radii). (●) represents the minimal typed structural vocabulary.

details how autoregressive LLMs inherently function as pointer networks to execute this symbolic routing.

3.1 Pointer-Centric Symbolic Representation

We formalize each data instance as a tuple $((\mathcal{G}, \mathcal{S}), \mathcal{M})$, where the input is explicitly decoupled into a geometric reference table $\mathcal{G}$ and an orthographic SVG sequence $\mathcal{S}$, while $\mathcal{M}$ represents the corresponding CAD command sequence. Our formulation is governed by a core design principle: continuous geometric parameters (e.g., spatial coordinates, radii) are instantiated exactly once in $\mathcal{G}$ and subsequently accessed via discrete references. Consequently, the input reference table $\mathcal{G}$ constructs a unified dictionary of unique geometric values, and both the input SVG sequence $\mathcal{S}$ and the target program $\mathcal{M}$ are synthesized by pointing to these indices rather than autoregressively generating raw numerical values.

As illustrated in Fig. 2, to facilitate seamless alignment, we map both the input orthographic SVGs $\mathcal{S}$ and the output CAD sequences $\mathcal{M}$ into a unified command vocabulary of geometric primitives (e.g., Line, Circle, and Arc). Such a shared representation space enforces a rigorous topological correspondence between 2D sketches and 3D construction logic. By bypassing intricate, hand-crafted geometric optimizations, this formulation leverages a scalable sequence-to-sequence modeling paradigm, aligning with the principles of general-purpose learning systems.

Unified Geometric Table. Let $\mathcal{V} = \{\text{Top, Front, Side}\}$ denote the set of orthographic views. We define a shared geometric table $\mathcal{G}$ consisting of reusable pointers: view specific point pointers $\mathcal{P} = \{T_i\}_{i \geq 1} \cup \{F_i\}_{i \geq 1} \cup \{S_i\}_{i \geq 1}$, length scalars $\mathcal{N} = \{N_i\}_{i \geq 1}$, and radius scalars $\mathcal{R} = \{R_i\}_{i \geq 1}$. Beyond pointers, the language incorporates a minimal typed structural vocabulary. This includes geometric primitives $\mathcal{E} = \{\text{L, C, A}\}$ (denoting Line, Circle, and Arc, respectively), program structure tokens $\mathcal{C} = \{\text{Sketch, Profile, Loop, Extrude}\}$, and Boolean operations $\mathcal{O} = \{\text{NewBody, Join, Cut}\}$. To preserve a fully discrete representation space, any necessary continuous quantities not captured by the pointer inventory (e.g., sketch plane offsets) are uniformly quantized into discrete tokens. This ensures the symbolic grammar remains closed without reverting to continuous numerical regression. We note that while verbose operation names are utilized throughout this paper for expository clarity, they are systematically abbreviated during actual training to minimize context length. Comprehensive details regarding this simplification protocol are provided in the Appendix B.

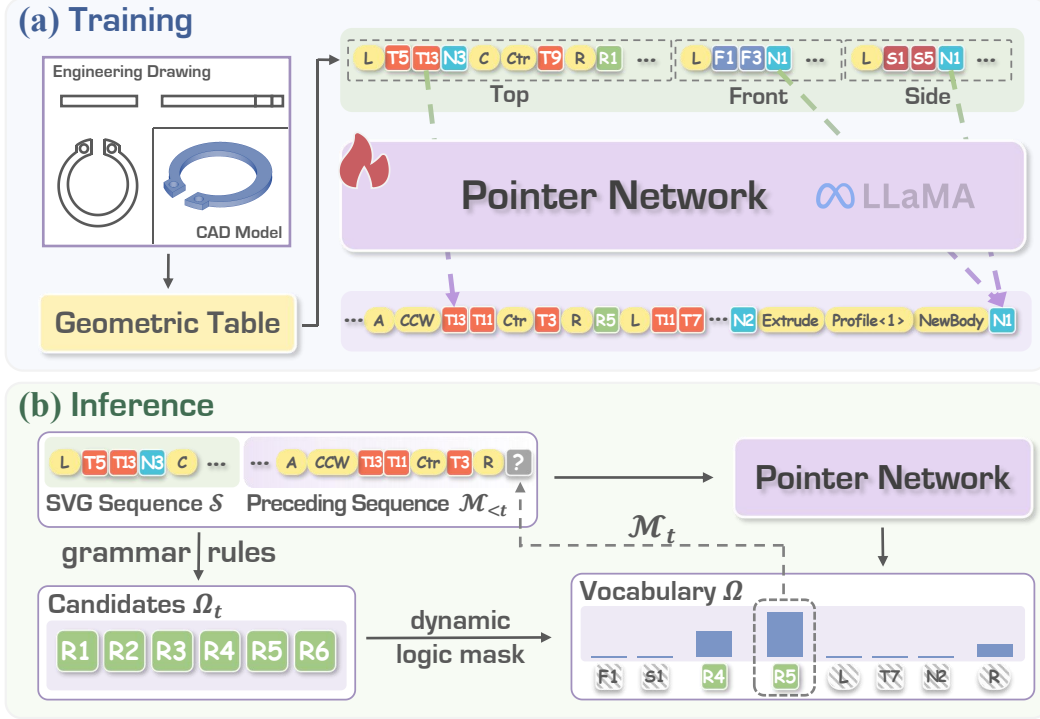


Figure 3: **Overview of the ForgeCAD framework.** (a) **Training Phase:** Both 2D engineering drawings and 3D target CAD models are parsed into a shared specific pointer notation. The LLM is then fine-tuned to autoregressively predict the target CAD sequence, utilizing its inherent self-attention as a pointer mechanism to retrieve discrete geometric tokens. (b) **Inference Phase:** At each decoding step t , valid candidate tokens are strictly constrained by programmatic grammar rules. This deterministic state machine applies dynamic logit masking to the vocabulary space, ensuring the generated sequence incrementally resolves into a topologically valid CAD program. Here, (Ⓢ) denotes the target token at step t , and (Ⓢ) indicates an invalid candidate eliminated by the logit mask.

Homogeneous Pointer Alignment. Under this vocabulary, the joint input context $(\mathcal{G}, \mathcal{S})$ acts as a serialized topological graph: $\mathcal{G}$ first defines the shared symbol inventory, followed by view-conditioned statements in $\mathcal{S}$ that bind these symbols to geometric primitives. To eliminate planar ambiguity, input arcs are explicitly parameterized with a midpoint p_m alongside their endpoints, center, and radius, denoted as $(p_1, p_m, p_2, p_c, r) \in \mathcal{P}^4 \times \mathcal{R}$. The output $\mathcal{M}$ is a procedural CAD program formulated within the same geometric domain. Unlike the input, output arcs are parameterized using an orientation token $d \in \{\text{CW}, \text{CCW}\}$ instead of a midpoint, i.e., $(d, p_1, p_2, p_c, r) \in \mathcal{D} \times \mathcal{P}^3 \times \mathcal{R}$. This procedural formulation is sufficient for deterministic 3D reconstruction while being more natural for program generation. Furthermore, extrusion commands seamlessly dictate 3D topology by combining high-level Boolean operations $o \in \mathcal{O}$ with signed distances derived directly from the scalar pointers $\mathcal{N}$.

Decoupling Geometry from Execution. This representation deliberately embeds the inductive bias of pointer networks into the token space. The model is not burdened with synthesizing exact geometric quantities *de novo*. Instead, it acts as a symbolic router: after reading a compact, typed inventory $\mathcal{G}$, it generates the CAD program $\mathcal{M}$ largely by selecting, reordering, and composing existing contextual symbols. By decoupling symbol declaration from structural execution, we significantly reduce the output entropy and bypass brittle spatial heuristics, allowing the model to learn robust cross-view geometric consistencies purely through scalable sequence modeling. Comprehensive details regarding the token representation, empirical analysis of this routing behavior, and concrete sequence examples are provided in Appendices B, E, and D.

3.2 Generative Pointer Routing via Sequence Modeling

Having reformulated CAD construction as a symbolic routing task, we now detail the modeling framework, summarized in Fig. 3. A key tenet of our approach is to eschew custom-designed architectural modifications (e.g., explicit copy mechanisms or specialized graph neural networks) in favor of the scalable, general-purpose sequence modeling capabilities of standard autoregressive LLMs.

Pointer Mechanism via LLMs. We hypothesize that the standard attention mechanism within Transformer architectures inherently functions as a powerful pointer network when conditioned on our proposed representation. As illustrated in Fig. 3(a), given the joint input context $(\mathcal{G}, \mathcal{S})$ and the previously generated sequence $\mathcal{M}_{<t}$, the model predicts the next token $\mathcal{M}_t$. When the target $\mathcal{M}_t$ is a geometric pointer (e.g., $T_i \in \mathcal{P}$ or $N_i \in \mathcal{N}$), the self-attention heads naturally learn to assign high attention scores to the corresponding symbol declarations within the context window. This allows the model to "point to" exact geometric identities from the reference table $\mathcal{G}$ without requiring a specialized attention routing module. By framing the problem purely within the discrete token space, we leverage the LLM’s pre-trained capacity for long-context dependency tracking to ensure accurate symbol retrieval and maintain strict procedural consistency throughout the CAD construction process.

Training Objective. We formulate the 3D CAD sequence generation as an autoregressive language modeling task over a heterogeneous vocabulary space. Let $\mathcal{V}_{\text{cmd}}$ denote the fixed procedural vocabulary (e.g., topological commands and Boolean operations), and $\mathcal{V}_{\text{ptr}}(\mathcal{G})$ represent the dynamic pointer vocabulary conditionally instantiated by the geometric table $\mathcal{G}$. The joint output vocabulary is thus defined as $\Omega = \mathcal{V}_{\text{cmd}} \cup \mathcal{V}_{\text{ptr}}(\mathcal{G})$.

To explicitly model the dual nature of this symbolic routing process, we decompose the standard Next-Token Prediction (NTP) objective. Given the joint input context and the generated history up to step t , denoted as $\mathcal{Z}_{<t} = (\mathcal{G}, \mathcal{S}, \mathcal{M}_{<t})$, the model parameters θ are optimized to minimize the following decoupled negative log-likelihood:

$$\mathcal{L}(\theta) = - \sum_{t=1}^{|\mathcal{M}|} \left[\underbrace{\mathbf{1}_{\{\mathcal{M}_t \in \mathcal{V}_{\text{cmd}}\}} \log P_{\theta}(\mathcal{M}_t \mid \mathcal{Z}_{<t})}_{\text{Procedural Structure Generation}} + \underbrace{\mathbf{1}_{\{\mathcal{M}_t \in \mathcal{V}_{\text{ptr}}(\mathcal{G})\}} \log P_{\theta}(\mathcal{M}_t \mid \mathcal{Z}_{<t})}_{\text{Symbolic Pointer Routing}} \right], \quad (1)$$

where $\mathbf{1}_{\{\cdot\}}$ denotes the indicator function. By mathematically offloading the continuous geometric parameters into the static inventory $\mathcal{V}_{\text{ptr}}(\mathcal{G})$, the second term explicitly optimizes the pointer mechanism. This forces the LLMs to accurately route to exact geometric identities, rather than penalizing infinitesimal deviations in coordinate regression. Consequently, this discrete, bipartite objective not only stabilizes the training dynamics and optimizes logical selection, but also effectively mitigates the spatial hallucinations inherently prevalent in numerical regression tasks.

Inference and Decoding. During inference, as depicted in Fig. 3(b), **ForgeCAD** autoregressively synthesizes the CAD program $\mathcal{M}$ token by token. To guarantee the syntactic validity of the generated procedural sequences, we strictly constrain the decoding process by injecting programmatic grammar rules directly into the inference pipeline. Specifically, at each decoding step t , a deterministic state machine evaluates the preceding sequence $\mathcal{M}_{<t}$ to yield a dynamically restricted candidate set $\Omega_t \subseteq \Omega$. The token selection under greedy decoding is formally defined as:

$$\mathcal{M}_t = \arg \max_{v \in \Omega_t} P_{\theta}(v \mid \mathcal{Z}_{<t}), \quad (2)$$

where $\mathcal{Z}_{<t} = (\mathcal{G}, \mathcal{S}, \mathcal{M}_{<t})$ represents the joint context. In practice, this constrained decoding is implemented via dynamic logit masking prior to the softmax operation:

$$\tilde{z}_t(v) = \begin{cases} z_t(v), & \text{if } v \in \Omega_t \\ -\infty, & \text{otherwise} \end{cases}, \quad (3)$$

where $z_t(v)$ denotes the raw pre-softmax logit emitted by the LLM for token v , and $\tilde{z}_t(v)$ is the masked logit. Tokens assigned a logit of $-\infty$ correspond to the masked candidates (⊞) illustrated in Fig. 3(b). By deterministically nullifying grammatically invalid transitions (e.g., restricting the token following a Center command strictly to a point pointer from $\mathcal{P}$), this constraint ensures that every

generated pointer sequence resolves into a fully compilable CAD program. A comprehensive list of the programmatic grammar rules applied during this decoding phase is detailed in Appendix C. Consequently, this guarantees a robust and deterministic 3D reconstruction driven purely by the referenced geometric table $\mathcal{G}$.

4 Experiments

4.1 Experimental Setup

Datasets. To maintain consistency with prior research [41], we evaluate our proposed model, **ForgeCAD**, on DeepCAD [55], a large-scale dataset of 3D sketch and extrude CAD construction sequences. The dataset comprises 178,238 sequences, which are randomly partitioned into training, validation, and test splits following a standard ratio of 90%, 5%, and 5%. Adhering to established preprocessing protocols [61, 60, 25, 31], we first filter out duplicate and topologically invalid sequences to ensure high data quality. Subsequently, we process the raw data to extract the unified geometric table $\mathcal{G}$ and serialize the orthographic SVGs $\mathcal{S}$ alongside the target CAD sequences $\mathcal{M}$. This transforms the dataset into the homogeneous, pointer-centric symbolic format detailed in Sec. 3.1.

Implementation Details. We instantiate **ForgeCAD** using LLaMA-3-7B [35] as the base LLM, chosen for its strong sequence modeling capabilities and competitive performance among open-source foundation models. To ensure parameter-efficient fine-tuning, we employ LoRA [14] with a rank of $r = 8$ and $\alpha = 32$. For the hardware setup, models with parameters up to 3B are trained on a cluster of eight NVIDIA RTX 4090 GPUs, while larger models are trained on eight NVIDIA L40 GPUs. We employ the AdamW optimizer [34] with a batch size of 16. We apply a cosine annealing learning rate schedule initialized at 5×10^{-4} and train the model for 15 epochs. During inference, we employ greedy decoding coupled with the dynamic logit masking strategy (i.e., grammar-constrained decoding) described in Sec. 3.2 to guarantee the syntactic validity of the generated procedural sequences.

Metrics. Our approach frames the translation of 2D engineering drawings into 3D CAD command sequences as a purely symbolic generation task; accordingly, we adopt a hierarchical evaluation protocol from prior CAD reconstruction literature [18, 41] to assess both procedural accuracy and geometric fidelity. To measure structural correctness, we compute the F1 score for the predicted extrusion operations and geometric primitive types, evaluating their exact occurrences within the command sequence. At the geometric level, the generated command sequences $\mathcal{M}$ are compiled via OpenCASCADE [1] into 3D boundary representations (B-Reps), which are then quantitatively compared against the ground-truth models using mean and median Chamfer Distances (CD) [8]. Finally, despite strict grammatical constraints, autoregressive token generation does not inherently guarantee geometric manifold closure. Consequently, we report the Invalidity Ratio (IR), which represents the percentage of predicted sequences that fail to successfully compile into valid 3D solid models.

4.2 Performance Comparison with Existing Methods

Baselines. We evaluate **ForgeCAD** against two primary baselines: Drawing2CAD [41], the current state-of-the-art method for CAD generation from vector drawings, and an adapted version of DeepCAD [55]. We utilize the official implementations for both methods to ensure a fair evaluation. Following the adaptation protocol established in prior literature [41], we modify the original DeepCAD architecture by substituting its standard CAD command input with an SVG command sequence, leaving all other configurations entirely unchanged.

Quantitative Results. In our quantitative evaluation, we compare our approach against two baseline methods, DeepCAD [55] and Drawing2CAD [41]. As demonstrated in Table 1, our proposed model, **ForgeCAD**, consistently outperforms both baselines across all evaluated metrics. This performance gap largely stems from the inability of the baseline models to explicitly establish a topological correspondence with the input engineering drawings. Instead of functioning as a symbolic router that efficiently queries a shared geometric memory, these baselines are forced to synthesize the complete

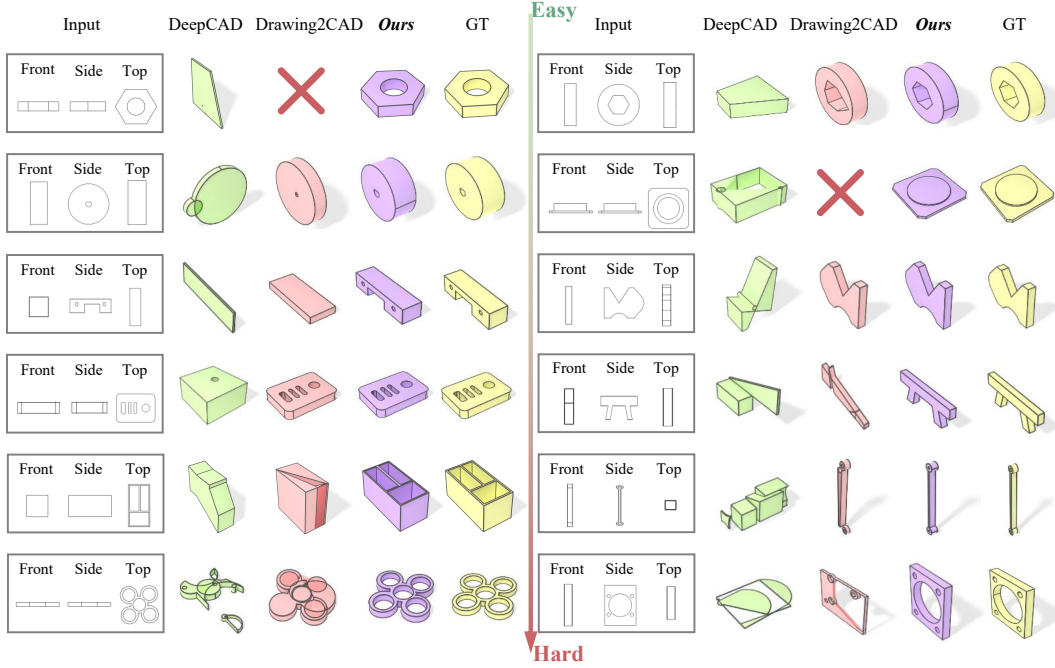


Figure 4: Qualitative comparison of 3D CAD generation from 2D engineering drawings on the DeepCAD dataset.

command sequence and precise numerical values entirely from scratch. Notably, the improvement in the F1 score is relatively modest compared to the gains in MeanCD and MedianCD. We attribute this to the inherent non-uniqueness of CAD command sequences, as sequence-level evaluations may not fully reflect the improvements captured by direct 3D geometric measurements. Overall, these results highlight the clear advantage of **ForgeCAD** in ensuring procedural validity, generation quality, and strict geometric fidelity to the engineering drawings.

Table 1: Performance comparison on the DeepCAD test set. MeanCD and MedianCD are multiplied by 10^2 . Best performances are in **bold**.

Model	F1 Score $\uparrow$				IR $\downarrow$	MeanCD $\downarrow$	MedianCD $\downarrow$
	Line	Arc	Circle	Extrude			
DeepCAD [55]	73.16	80.53	81.20	71.37	24.10%	4.05	0.94
Drawing2CAD [41]	79.37	82.11	82.83	75.15	20.48%	3.77	0.86
ForgeCAD (ours)	88.79	89.39	93.00	84.61	7.82 %	2.57	0.47

Qualitative Results. In our qualitative analysis, as illustrated in Fig. 4, **ForgeCAD** generates CAD models that capture the precise geometric specifications detailed in the engineering drawings, demonstrating clear improvements over the baseline models. Notably, as model complexity increases, DeepCAD [55] and Drawing2CAD [41] frequently produce invalid geometries, open boundary representations, and inaccurate spatial features including misaligned holes and incorrect topological positions. Conversely, our approach maintains highly stable generation capabilities even when tackling intricate designs. This robustness is directly attributable to our pointer formulation. In complex engineering drawings characterized by dense and closely spaced geometric entities, baseline models often conflate adjacent lines and vertices, causing severe spatial confusion. By explicitly anchoring each geometric primitive to a unique index in the shared geometric memory $\mathcal{G}$, **ForgeCAD** distinctly resolves these visually entangled features and faithfully preserves subtle structural details.

4.3 Ablation Studies

Table 2: Ablation study on different LLMs and fine-tuning strategies. Transformer-4M is a lightweight Transformer-based [47] baseline. ‘w/o Pre-trained’ and ‘w/ Pre-trained’ denote training from scratch and initializing with pre-trained weights. **Bold** and underline denote the best and second-best results, respectively.

Model	IR↓	MeanCD↓	MedianCD↓
Transformer-4M (w/o Pre-trained, Full)	11.13 %	3.41	0.82
LLaMA3.2-3B (w/ Pre-trained, LoRA)	<u>9.87 %</u>	<u>2.99</u>	<u>0.52</u>
LLaMA3-8B (w/ Pre-trained, LoRA)	7.82 %	2.57	0.47

Table 3: Effectiveness analysis of the Pointer Mechanism. ‘w/o Pointer’ denotes directly generating the numeric CAD command sequence based on the input engineering drawings. **Bold** and underline denote the best and second-best results, respectively.

	IR ↓	MeanCD ↓	MedianCD ↓
LLaMA3.2-3B (w/ Pointer)	<u>9.87%</u>	<u>2.99</u>	<u>0.52</u>
LLaMA3.2-3B (w/o Pointer)	15.56%	3.43	0.71
LLaMA3-8B (w/ Pointer)	7.82%	2.57	0.47
LLaMA3-8B (w/o Pointer)	10.19%	3.04	0.61

Ablation of LLM Configurations. As shown in Table 2, the poor performance of the Transformer-4M model suggests that pre-trained weights are indispensable for this task. Beyond the necessity of pre-training, the results exhibit a clear positive correlation between model scale and routing capability; notably, scaling the backbone from 3B to 8B parameters yields consistent improvements across all metrics, culminating in the lowest Invalidity Ratio (7.82%).

Effectiveness of the Pointer-Centric Symbolic Representation. As shown in Table 3, without the pointer mechanism, the model exhibits substantially degraded metrics across all evaluations, underscoring the vital necessity of our proposed representation. In essence, the Pointer Mechanism helps the model to focus on the generation of structural topology and procedural logic, rather than struggling with numerical regression. By delegating the exact coordinate values to the unified geometric memory $\mathcal{G}$, we effectively mitigate the numerical hallucination commonly observed in standard language models, evidenced by the significant drop in both Chamfer Distances and Invalidity Ratio.

Effectiveness of Deterministic State Machine Decoding.

The ablation of the Deterministic State Machine Decoding (DSM), as detailed in Table 4, leads to the decline in performance metrics (IR, MeanCD, and MedianCD), confirming the necessity of structurally constrained generation. It is worth noting that the base LLM under unconstrained vanilla decoding still rarely deviates from valid topologies, largely owing to the strong capacity provided by our shared geometric memory. Nevertheless, autoregressive generation without explicit constraints is inherently vulnerable to edge-case hallucinations. DSM eliminates this uncertainty by enforcing strict syntactic rules at each decoding step. By providing both standard greedy decoding and DSM, **ForgeCAD** introduces a flexible precision-efficiency trade-off: users can opt for a $3\times$ faster inference speed for rapid iterations, or activate DSM to ensure guaranteed topological correctness for rigorous engineering applications.

Table 4: Effectiveness of Deterministic State Machine Decoding (DSM). Base LLMs are LLaMA3.2-3B. **Bold** denote the best results.

	IR ↓	MeanCD ↓	MedianCD ↓
w/ DSM	9.87%	2.99	0.52
w/o DSM	10.31%	3.12	0.59

5 Conclusion

In this paper, we present **ForgeCAD**, a novel framework that reformulates 2D-to-3D CAD generation as a symbolic routing task. By decoupling geometric parameters into a unified reference table, our pointer-centric representation enables standard autoregressive LLMs to inherently function as

highly accurate pointer networks. Extensive qualitative and quantitative evaluations demonstrate that **ForgeCAD** improves generation quality, validity, and fidelity to the engineering drawings. Ultimately, this work highlights the profound potential of scalable, data-driven approaches for precise topological and geometric alignment.

References

- [1] Open cascade. <https://dev.opencascade.org/>.
- [2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [3] K. Alrashedy, P. Tambwekar, Z. H. Zaidi, M. Langwasser, W. Xu, and M. C. Gombolay. Generating CAD code with vision-language models for 3d designs. In *ICLR*, 2025.
- [4] J. D. Camba, P. Company, and F. Naya. Sketch-based modeling in mechanical engineering design: Current status and opportunities. *Computer-Aided Design*, 2022.
- [5] C. Chen, J. Wei, T. Chen, C. Zhang, X. Yang, S. Zhang, B. Yang, C.-S. Foo, G. Lin, Q. Huang, et al. Cadrafter: Generating computer-aided design models from unconstrained images. In *CVPR*, 2025.
- [6] B. C. Das, M. H. Amini, and Y. Wu. Security and privacy challenges of large language models: A survey. *ACM Computing Surveys*, 2025.
- [7] Y. Dong, J. Ding, X. Jiang, G. Li, Z. Li, and Z. Jin. Codescore: Evaluating code generation by learning code execution. *ACM Transactions on Software Engineering and Methodology*, 2025.
- [8] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017.
- [9] R. Fan, F. He, Y. Liu, and J. Lin. A history-based parametric cad sketch dataset with advanced engineering commands. *Computer-Aided Design*, 2025.
- [10] R. Furferi, L. Governì, M. Palai, Y. Volpe, et al. From 2d orthographic views to 3d pseudo-wireframe: An automatic procedure. *International Journal of Computer Applications*, 2010.
- [11] J.-H. Gong, G.-F. Zhang, H. Zhang, and J.-G. Sun. Reconstruction of 3d curvilinear wire-frame from three orthographic views. *Computers & Graphics*, 2006.
- [12] X. Gu, M. Chen, Y. Lin, Y. Hu, H. Zhang, C. Wan, Z. Wei, Y. Xu, and J. Wang. On the effectiveness of large language models in domain-specific code generation. *ACM Transactions on Software Engineering and Methodology*, 2025.
- [13] A. B. Harish and A. R. Prasad. Photo2cad: Automated 3d solid reconstruction from 2d drawings using opencv. *arXiv preprint arXiv:2101.04248*, 2021.
- [14] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 2022.
- [15] W. Hu, J. Zheng, Z. Zhang, X. Yuan, J. Yin, and Z. Zhou. Plankassembly: Robust 3d reconstruction from three orthographic views with learnt shape programs. In *ICCV*, 2023.
- [16] B. Jiang, X. Chen, W. Liu, J. Yu, G. Yu, and T. Chen. Motiongpt: Human motion as a foreign language. *NeurIPS*, 2023.
- [17] B. Kerbl, G. Kopanas, T. Leimkühler, G. Drettakis, et al. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 2023.
- [18] M. S. Khan, E. Dupont, S. A. Ali, K. Cherenkova, A. Kacem, and D. Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *CVPR*, 2024.
- [19] M. S. Khan, M. Usama, R. A. Potamias, D. Stricker, M. Z. Afzal, J. Deng, and I. Elezi. Dreamcad: Scaling multi-modal cad generation using differentiable parametric surfaces. *arXiv preprint arXiv:2603.05607*, 2026.
- [20] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. Abc: A big cad model dataset for geometric deep learning. In *CVPR*, 2019.

- [21] J. Kuang, Y. Shen, J. Xie, H. Luo, Z. Xu, R. Li, Y. Li, X. Cheng, X. Lin, and Y. Han. Natural language understanding and inference with mllm in visual question answering: A survey. *ACM Computing Surveys*, 2025.
- [22] M.-H. Kuo. Reconstruction of quadric surface solids from three-view engineering drawings. *Computer-Aided Design*, 1998.
- [23] Z. Lai, Y. Zhao, H. Liu, Z. Zhao, Q. Lin, H. Shi, X. Yang, M. Yang, S. Yang, Y. Feng, et al. Hunyuan3d 2.5: Towards high-fidelity 3d assets generation with ultimate details. *arXiv preprint arXiv:2506.16504*, 2025.
- [24] M. Lee, D. Zhang, C. Jambon, and Y. M. Kim. Brepdiff: Single-stage b-rep diffusion model. In *SIGGRAPH*, 2025.
- [25] J. Li, Y. Fu, and F. Chen. Dtgbreppen: A novel b-rep generative model through decoupling topology and geometry. In *CVPR*, 2025.
- [26] J. Li, K. Pan, Z. Ge, M. Gao, W. Ji, W. Zhang, T. Chua, S. Tang, H. Zhang, and Y. Zhuang. Fine-tuning multimodal llms to follow zero-shot demonstrative instructions. In *ICLR*, 2024.
- [27] P. Li, W. Zhang, J. Chen, and D. Yan. Stitch-a-shape: Bottom-up learning for b-rep generation. In *SIGGRAPH*, 2025.
- [28] P. Li, W. Zhang, J. Guo, J. Chen, and D.-M. Yan. Revisiting cad model generation by learning raster sketch. In *AAAI*, 2025.
- [29] P. Li, W. Zhang, W. Quan, B. Zhang, P. Wonka, and D. Yan. Brepopt: Autoregressive b-rep generation with voronoi half-patch. *TOG*, 2025.
- [30] Y. Li, C. Lin, Y. Liu, X. Long, C. Zhang, N. Wang, X. Li, W. Wang, and X. Guo. Caddreamer: Cad object generation from single-view images. In *CVPR*, 2025.
- [31] S. Liu, W. Chen, and A. Wu. Hidigen: Hierarchical diffusion for b-rep generation with explicit topological constraints. *arXiv preprint arXiv:2604.02847*, 2026.
- [32] S.-X. Liu, S.-M. Hu, Y.-J. Chen, and J.-G. Sun. Reconstruction of curved solids from engineering drawings. *Computer-Aided Design*, 2001.
- [33] Y. Liu, D. Xu, X. Yu, X. Xu, D. Cohen-Or, H. Zhang, and H. Huang. Hola: B-rep generation using a holistic latent representation. *TOG*, 2025.
- [34] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.
- [35] A. Meta. Introducing meta llama 3: The most capable openly available llm to date. *Meta AI*, 2024.
- [36] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 2021.
- [37] C. Nash, Y. Ganin, S. A. Eslami, and P. Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *ICML*, 2020.
- [38] A. Nichol, H. Jun, P. Dhariwal, P. Mishkin, and M. Chen. Point-e: A system for generating 3d point clouds from complex prompts. *arXiv preprint arXiv:2212.08751*, 2022.
- [39] S. Ohsuga. Toward intelligent cad systems. *Computer-Aided Design*, 1989.
- [40] I. Puhachov, C. Martens, P. G. Kry, and M. Bessmeltsev. Reconstruction of machine-made shapes from bitmap sketches. *TOG*, 2023.
- [41] F. Qin, S. Lu, J. Hou, C. Wang, M. Fang, and L. Liu. Drawing2cad: Sequence-to-sequence learning for cad generation from vector drawings. In *ACM MM*, 2025.
- [42] D. Rukhovich, E. Dupont, D. Mallis, K. Cherenkova, A. Kacem, and D. Aouada. Cad-recode: Reverse engineering cad code from point clouds. In *ICCV*, 2025.
- [43] P. Shojaei, K. Meidani, S. Gupta, A. B. Farimani, and C. K. Reddy. LLM-SR: scientific equation discovery via programming with large language models. In *ICLR*, 2025.
- [44] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, M. Amin, L. Hou, K. Clark, S. R. Pfohl, H. Cole-Lewis, et al. Toward expert-level medical question answering with large language models. *Nature medicine*, 2025.

- [45] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [46] M. Tovey. Drawing and cad in industrial design. *Design Studies*, 1989.
- [47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *NeurIPS*, 2017.
- [48] S. Wang, C. Chen, X. Le, Q. Xu, L. Xu, Y. Zhang, and J. Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. In *AAAI*, 2025.
- [49] S. Wang, C. Chen, X. Le, Q. Xu, L. Xu, Y. Zhang, and J. Yang. Cad-gpt: Synthesising cad construction sequence with spatial reasoning-enhanced multimodal llms. In *AAAI*, 2025.
- [50] S. Wang, C. Chen, X. Le, Q. Xu, L. Xu, Y. Zhang, and J. Yang. CAD-GPT: synthesising CAD construction sequence with spatial reasoning-enhanced multimodal llms. In *AAAI*, 2025.
- [51] W. Wang and G. G. Grinstein. A survey of 3d solid reconstruction from 2d projection line drawings. In *Computer Graphics Forum*. Wiley Online Library, 1993.
- [52] X. Wang, J. Zheng, Y. Hu, H. Zhu, Q. Yu, and Z. Zhou. From 2d cad drawings to 3d parametric models: A vision-language approach. In *AAAI*, 2025.
- [53] K. D. Willis, Y. Pu, J. Luo, H. Chu, T. Du, J. G. Lambourne, A. Solar-Lezama, and W. Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *TOG*, 2021.
- [54] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *NeurIPS*, 2016.
- [55] R. Wu, C. Xiao, and C. Zheng. Deepcad: A deep generative network for computer-aided design models. In *ICCV*, 2021.
- [56] S. Wu, A. H. Khasahmadi, M. Katz, P. K. Jayaraman, Y. Pu, K. Willis, and B. Liu. Cadvlm: Bridging language and vision in the generation of parametric cad sketches. In *ECCV*. Springer, 2024.
- [57] J. Xiang, Z. Lv, S. Xu, Y. Deng, R. Wang, B. Zhang, D. Chen, X. Tong, and J. Yang. Structured 3d latents for scalable and versatile 3d generation. In *CVPR*, 2025.
- [58] J. Xu, C. Wang, Z. Zhao, W. Liu, Y. Ma, and S. Gao. Cad-mlm: Unifying multimodality-conditioned cad generation with mllm. *arXiv preprint arXiv:2411.04954*, 2024.
- [59] X. Xu, P. Jayaraman, J. Lambourne, Y. Liu, D. Malpure, and P. Meltzer. Autobrep: Autoregressive b-rep generation with unified topology and geometry. In *SIGGRAPH*, 2025.
- [60] X. Xu, J. Lambourne, P. Jayaraman, Z. Wang, K. Willis, and Y. Furukawa. Brep-gen: A b-rep generative diffusion model with structured latent geometry. *TOG*, 2024.
- [61] X. Xu, K. D. D. Willis, J. G. Lambourne, C. Cheng, P. K. Jayaraman, and Y. Furukawa. Skexgen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *ICML*, 2022.
- [62] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- [63] M. Yavartanoo, S. Hong, R. Neshatavar, and K. M. Lee. Text2cad: Text to 3d cad generation via technical drawings. *arXiv preprint arXiv:2411.06206*, 2024.
- [64] Y. Yuan, S. Sun, Q. Liu, and J. Bian. Cad-editor: A locate-then-infill framework with automated training data synthesis for text-based CAD editing. In *ICML*, 2025.
- [65] C. Zhang, R. Piquié, A. Polette, G. Carasi, H. De Charnace, and J.-P. Pernot. Automatic 3d cad models reconstruction from 2d orthographic drawings. *Computers & Graphics*, 2023.
- [66] C. Zhang, A. Polette, R. Piquié, G. Carasi, H. De Charnace, and J.-P. Pernot. ecad-net: Editable parametric cad models reconstruction from dumb b-rep models using deep neural networks. *Computer-Aided Design*, 2025.
- [67] R. Zhang, J. Han, C. Liu, A. Zhou, P. Lu, Y. Qiao, H. Li, and P. Gao. Llama-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *ICLR*, 2024.

- [68] Y. Zhang, F. Feng, J. Zhang, K. Bao, Q. Wang, and X. He. Collm: Integrating collaborative embeddings into large language models for recommendation. *TKDE*, 2025.
- [69] Z. Zhang, k. liu, J. Liu, W. Wang, B. Lin, L. Xie, C. Shen, and D. Cai. Geocad: Local geometry-controllable cad generation with large language models. In *NeurIPS*. Curran Associates, Inc., 2025.
- [70] Z. Zhang, S. Sun, W. Wang, D. Cai, and J. Bian. Flexcad: Unified and versatile controllable CAD generation with fine-tuned large language models. In *ICLR*, 2025.
- [71] L. Zhou, Y. Du, and J. Wu. 3d shape generation and completion through point-voxel diffusion. In *ICCV*, 2021.

Appendix

Due to space limitations in the main paper, we provide additional results and discussions in this appendix, organized as follows:

- Sec. A Details on Data Processing Pipeline
- Sec. B Details on Sequence Simplification Protocol
- Sec. C Details on Deterministic State Machine Decoding
- Sec. D Examples of Constructed Drawings and Command Sequences
- Sec. E Analysis of the Symbolic Routing Mechanism
- Sec. F Equivariance under Symbolic Renaming
- Sec. G Additional Qualitative Results
- Sec. H Failure Cases, Limitations and Future Work

A Details on Data Processing Pipeline

For each CAD model, the input representation is produced by a kernel-level orthographic projection pipeline that closely matches the rendering workflow of modern CAD systems. As illustrated in Algorithm. 1, the model is first normalized into a canonical 3D coordinate frame, after which top, front, and side views are generated through hidden-line-removal rendering directly on the geometric kernel. Primitive entities extracted from these projections are organized into a geometric reference table $\mathcal{G}$ and an orthographic SVG sequence $\mathcal{S}$, while the corresponding construction program is serialized as the target CAD command sequence $\mathcal{M}$. Hence, each data instance is formulated as $((\mathcal{G}, \mathcal{S}), \mathcal{M})$, with an input representation that is intrinsically aligned with modern OpenCascade [1] projection pipelines.

B Details on Sequence Simplification Protocol

This section elaborates on the representation of engineering drawings and CAD command sequences. As summarized in Tables 5 and 6, we define structural representation for both domains, presenting each token in both a verbose and a compressed format. The verbose format provides an explicit, human-readable structure that facilitates the inspection of underlying geometric and CAD semantics. In contrast, the compressed format strips redundant keywords and abbreviates repeated markers to optimize sequence modeling efficiency. During training, the LLM operates exclusively on the compressed sequences for both input and prediction. Notably, by adopting this compressed formulation, we effectively reduce the training context length from 10K to 3K tokens.

C Details on Deterministic State Machine Decoding

During inference, we apply a finite-state constrained decoding procedure to ensure that the generated CAD command sequence satisfies the grammar in Table 6. At each decoding step, we append the decoded string of each candidate tokenizer token to the current output prefix and test whether the resulting string remains a valid grammar prefix. Candidate tokens that violate any rule are assigned a logit of $-\infty$ before greedy decoding.

1. **Sequence initialization.** Each sequence must start with a sketch declaration:

$$S \text{ <sid> <V> B \text{ <cq> D \text{ <sgn>,}$$

where $V \in \{T, F, S\}$, $cq \in [0, 63]$, and $sgn \in \{-1, 1\}$.

2. **Sketch indexing.** Sketch identifiers are globally monotonic. If m sketches have been completed, the next sketch must use $\text{<sid>} = m + 1$.
3. **Profile declaration.** A profile is declared as

$$P \text{ <k>.$$

Profile identifiers are also globally monotonic over the entire sequence, rather than being reset within each sketch.

Table 5: Engineering Drawings token representations. The table is divided into two sections: the top section defines the symbols for the unified geometry table, while the bottom section outlines the primitives for the engineering drawing table. The Geom (verbose) and G (compressed) prefixes denote that the followed tokens are used to represent an SVG primitive.

Type	Verbose Arguments	Compressed Arguments	Range	Description
Point	Point <pid> <xq> <yq>	P <pid> <xq> <yq>	$pid \in \{T_i, F_i, S_i\}$	Shared quantized point symbol.
Length	Length <lid> <lq>	N <lid> <lq>	$lq \in [0, 63]$	Shared quantized length symbol.
Radius	Radius <rid> <rq>	R <rid> <rq>	$rq \in [0, 63]$	Shared quantized radius symbol.
Line	Geom <View> Line <p1> <p2> <lid>	G <V> L <p1> <p2> <lid>	$V \in \{T, F, S\}$	Pointer-based line primitive.
Circle	Geom <View> Circle Center <pc> Radius <rid>	G <V> C <pc> <rid>	$rid = R_i$	Pointer-based circle primitive.
Arc	Geom <V> Arc <p1> <pm> <p2> C <pc> R <rid>	G <V> A <p1> <p2> C <pc> R <rid>+	$p, pc \in \{P_i\}$	Pointer-based arc primitive.

Table 6: CAD command sequence token representation.

Type	Verbose Arguments	Compressed Arguments	Range	Description
Sketch	Sketch <sid> View <V> Offset <cq> S <sgn>	S <sid> <V> B <cq> D <sgn>	$sgn \in \pm 1$	Sketch-plane declaration.
Profile	Profile <k>	P <k>	$k \geq 1$	Profile declaration.
Loop	Loop CCW / CW	L + / -	$+/-$	Loop orientation.
Line	Line <p1> <p2> Length <lid>	E <p1> <p2> <lid>	$p1, p2 \in \{P_i\}$	Pointer-based line primitive.
Circle	Circle Center <pc> Radius <rid>	O <pc> <rid>	$rid = R_i$	Pointer-based circle primitive.
Arc	Arc <dir> <p1> <p2> C <pc> R <rid>	A <d> <p1> <p2> C <pc> R <rid>	$d \in \{+, -\}$	Pointer-based arc primitive.
Extrude	Extrude <Op> Dist <dref> Prof <p1> <p2>	X <op> <dref> P <p1> <p2>	$op \in \{N, C, J\}$	Extrusion command.

Range note. All quantized values use 6-bit bins, i.e., integers in $[0, 63]$. Coordinate bins map to normalized coordinates in $[-1, 1]$, while length, radius, and distance bins map to normalized values in $[0, 2]$. The distance token $<dref>$ may be a shared symbol (N_i), a signed shared symbol ($-N_i$), or a quantized fallback value (Bin b or $-Bin$ b).

4. **Profile ownership.** Each profile is associated with the most recently declared sketch. All subsequent loop and primitive commands are assigned to the current profile until a new profile or sketch is opened.
5. **Loop declaration.** A loop can only be declared after a valid profile:

$$L + \quad \text{or} \quad L -.$$

In the current dataset, valid generated loops use the counter-clockwise token +; therefore, our decoder permits + by default.

Algorithm 1: *Construction of training instances from a CAD model.*

Data: CAD model $\mathcal{B}$ and its construction sequence $\mathcal{M}$

Result: Data instance $((\mathcal{G}, \mathcal{S}), \mathcal{M})$

```
1  $(\mathbf{c}, s) \leftarrow \text{normalize\_3D}(\mathcal{B})$ ; // Compute canonical 3D center and scale
2 foreach  $v \in \{\text{Top, Front, Side}\}$  do
3    $\mathcal{H}_v \leftarrow \text{HLR\_project}(\mathcal{B}, v)$ ; // Kernel-level orthographic projection with hidden-line
   removal
4    $\mathcal{P}_v \leftarrow \text{extract\_primitives}(\mathcal{H}_v)$ ; // Extract visible geometric primitives: lines, circles, and
   arcs
5    $\mathcal{P}_v \leftarrow \text{normalize\_to\_view}(\mathcal{P}_v, \mathbf{c}, s)$ ; // Map projected geometry into the normalized 2D
   coordinate frame
6    $\mathcal{P}_v \leftarrow \text{merge\_and\_deduplicate}(\mathcal{P}_v)$ ; // Remove repeated or overlapping primitives
7 end
8  $\mathcal{G} \leftarrow \text{build\_reference\_table}(\mathcal{P}_{\text{Top}}, \mathcal{P}_{\text{Front}}, \mathcal{P}_{\text{Side}})$ ; // Build shared geometric table, e.g., points,
   lengths, and radii
9  $\mathcal{S} \leftarrow \text{serialize\_views}(\mathcal{G}, \{\mathcal{P}_v\})$ ; // Convert the three orthographic views into the SVG
   sequence
10  $\mathcal{M} \leftarrow \text{serialize\_cad\_sequence}(\mathcal{M}, \mathcal{G})$ ; // Convert the CAD construction history into the target
   command sequence
11 return  $((\mathcal{G}, \mathcal{S}), \mathcal{M})$ 
```

6. **Primitive commands.** Geometric primitives can only be generated inside an active loop. The valid primitive forms are:

E $\langle p1 \rangle \langle p2 \rangle \langle lid \rangle$,
O $\langle pc \rangle \langle rid \rangle$,
A $\langle d \rangle \langle p1 \rangle \langle p2 \rangle$ C $\langle pc \rangle$ R $\langle rid \rangle$,

where $d \in \{+, -\}$.

7. **Point pointer validity.** Each point reference $\langle p1 \rangle$, $\langle p2 \rangle$, or $\langle pc \rangle$ must be a valid point symbol from the input point table. When the current sketch view is known, the decoder further restricts point candidates to the corresponding view prefix. For example, in a front-view sketch, only F-prefixed point symbols are allowed.

8. **Length pointer validity.** The line length token $\langle lid \rangle$ in

E $\langle p1 \rangle \langle p2 \rangle \langle lid \rangle$

must be a valid shared length symbol N_i from the input length table.

9. **Radius pointer validity.** The circle and arc radius token $\langle rid \rangle$ must be a valid shared radius symbol R_i from the input radius table.

10. **Command transition constraints.** The finite-state decoder enforces the following high-level transitions:

S $\rightarrow$ P,
P $\rightarrow$ L,
L $\rightarrow$ E | O | A,
E/O/A $\rightarrow$ E | O | A | L | P | S | X,
X $\rightarrow$ X | EOS.

Thus, extrusion commands cannot appear before a valid sketch–profile–loop structure has been generated.

11. **Extrusion command.** An extrusion command must have the form

X $\langle op \rangle \langle dref \rangle$ P $\langle p1 \rangle \langle p2 \rangle \dots$,

where $op \in \{N, C, J\}$ denotes new-body, cut, and join operations, respectively.

12. **Extrusion distance validity.** The distance reference $\langle dref \rangle$ must be one of the following:

N_i , $-N_i$, Bin b , $-\text{Bin } b$,

where N_i is a shared length symbol and $b \in [0, 63]$ is a quantized 6-bit bin.

13. **Profile reference validity in extrusion.** Each profile index referenced after the P marker in an extrusion command must correspond to a previously declared profile. Repeated profile references are allowed, since they occur in the training data.
14. **Extrusion-phase constraint.** Once the decoder enters the extrusion phase, it cannot return to sketch, profile, loop, or primitive generation. It may only generate additional extrusion commands or terminate.
15. **Termination rule.** The end-of-sequence token is permitted only after at least one complete extrusion command has been generated. The decoder does not allow termination from an incomplete command prefix.
16. **Line-boundary constraint.** A newline token is valid only after a complete command line. For example,

$S \backslash n$

is invalid because the sketch declaration is incomplete, whereas

$S \ 1 \ F \ B \ 52 \ D \ -1 \backslash n$

is valid. The decoder also disallows leading whitespace at the beginning of a new command line.

17. **Tokenizer-level masking.** Constraints are applied at the tokenizer-token level rather than at the symbolic CAD-token level. For each tokenizer candidate z , the decoder checks whether

$$y_{1:t} \oplus \text{decode}(z)$$

is still a valid CAD prefix, where $y_{1:t}$ is the current decoded output prefix and $\oplus$ denotes string concatenation. This handles tokenizer tokens that contain spaces, newlines, or multiple CAD symbols. All invalid candidates are masked by setting their logits to $-\infty$.

D Examples of Constructed Drawings and Command Sequences

To better illustrate the implementation details of unified geometry table and pointer-centric symbolic representation in main paper, we present some examples of constructed engineering drawings and cad command sequence, as shown in Fig. A1.

E Analysis of the Symbolic Routing Mechanism

To further examine whether the model learns a symbolic routing behavior, we analyze the next-token distribution at pointer-valued CAD slots (i.e., tokens where $\mathcal{M}_t \in \mathcal{V}_{\text{ptr}}(\mathcal{G})$) under teacher forcing. For each target sequence, we feed the ground-truth causal history $\mathcal{M}_{<t}$ to the model and compute the entropy, top-1 probability, and target-token probability of the next-token distribution. A lower entropy and a higher top-1 probability indicate that the model assigns a more concentrated distribution to the next symbolic reference, which is expected if the model performs pointer-like selection over the input symbol table $\mathcal{G}$.

Table 7 reports the aggregate statistics on test samples. The pointer-routing formulation produces highly concentrated distributions at symbolic reference positions. In particular, length and radius pointers exhibit very low entropy and high top-1 probabilities, suggesting that the model can confidently route CAD commands to reusable symbolic entities rather than regressing raw geometric values. Point pointers are relatively more challenging because they involve larger candidate sets and possible ordering ambiguity in sketch loops, but they still maintain a high top-1 probability.

Fig. A2 provides a qualitative example of this behavior. In the shown CAD object, two circle entities are highlighted, and the corresponding radius/center selection steps are visualized with the model’s top-1 prediction and probability distribution over alternative candidate tokens. The correct circle-related pointer receives the dominant probability mass, while other candidate tokens receive substantially lower probabilities. This example illustrates the slot-conditioned nature of the learned routing process: when the output command requires a circle reference, the model concentrates its distribution on the appropriate symbolic token from the input structure.

Together, the quantitative results in Table 7 and the case study in Fig. A2 indicate that the LLM does not merely perform unconstrained next-token generation. Instead, at CAD reference slots, its

Example 1

Instruction:

Generate the CAD sequence based on the input SVG sequence.##Input:\nP F1 2 29\nP F2 61 29\nP F3 2 34\nP F4 61 34\nP S1 22 29\nP S2 41 29\nP S3 22 34\nP S4 41 34\nP T1 2 22\nP T2 61 22\nP T3 2 41\nP T4 61 41\nN N1 4\nN N2 19\nN N3 60\nG T L T1 T2 N3\nG T L T1 T3 N2\nG T L T2 T4 N2\nG T L T3 T4 N3\nG F L F1 F2 N3\nG F L F1 F3 N1\nG F L F2 F4 N1\nG F L F3 F4 N3\nG S L S1 S2 N2\nG S L S1 S3 N1\nG S L S2 S4 N1\nG S L S3 S4 N2\n##Output:\n

Answer:

S 1 T B 29 D 1\nP 1\nL +\nE T1 T2 N3\nE T2 T4 N2\nE T3 T4 N3\nE T1 T3 N2\nX N N1 P 1

Example 2

Instruction:

Generate the CAD sequence based on the input SVG sequence.##Input:\nP F1 9 30\nP F2 17 30\nP F3 46 30\nP F4 54 30\nP F5 9 33\nP F6 17 33\nP F7 46 33\nP F8 54 33\nP S1 10 30\nP S2 37 30\nP S3 53 30\nP S4 10 33\nP S5 37 33\nP S6 53 33\nP T1 17 10\nP T2 46 10\nP T3 24 21\nP T4 39 21\nP T5 20 34\nP T6 43 34\nP T7 9 37\nP T8 54 37\nP T9 32 42\nP T10 32 53\nN N1 2\nN N2 14\nN N3 28\nN N4 43\nN N5 46\nG T L T1 T2 N3\nG T L T2 T8 N3\nG T L T3 T4 N2\nG T L T4 T6 N2\nG T L T5 T3 N2\nG T L T5 T9 N2\nG T L T7 T1 N3\nG T L T7 T10 N3\nG T L T9 T6 N2\nG T L T10 T8 N3\nG F L F1 F4 N5\nG F L F1 F5 N1\nG F L F2 F6 N1\nG F L F3 F7 N1\nG F L F4 F8 N1\nG F L F5 F8 N5\nG S L S1 S3 N4\nG S L S1 S4 N1\nG S L S2 S5 N1\nG S L S3 S6 N1\nG S L S4 S6 N4\n##Output:\n

Answer:

S 1 T B 30 D 1\nP 1\nL +\nE T3 T4 N2\nE T4 T6 N2\nE T6 T9 N2\nE T9 T5 N2\nE T3 T5 N2\nL +\nE T10 T7 N3\nE T7 T1 N3\nE T1 T2 N3\nE T2 T8 N3\nE T10 T8 N3\nX N N1 P 1\nX J N1 P 1

Example 3

Instruction:

Generate the CAD sequence based on the input SVG sequence.##Input:\nP F1 32 32\nP S1 29 9\nP S2 34 9\nP S3 29 54\nP S4 34 54\nP T1 9 29\nP T2 54 29\nP T3 9 34\nP T4 54 34\nN N1 4\nN N2 44\nR R1 22\nG T L T1 T3 N1\nG T L T1 T2 N2\nG T L T2 T4 N1\nG T L T3 T4 N2\nG F C F1 R1\nG S L S1 S2 N1\nG S L S3 S4 N1\nG S L S1 S3 N2\nG S L S2 S4 N2\n##Output:\n

Answer:

S 1 F B 34 D -1\nP 1\nL +\nO F1 R1\nX N N1 P 1

Example 4

Instruction:

Generate the CAD sequence based on the input SVG sequence.##Input:\nP F1 25 10\nP F2 38 10\nP F3 25 53\nP F4 38 53\nP S1 32 32\nP T1 25 10\nP T2 38 10\nP T3 25 53\nP T4 38 53\nN N1 12\nN N2 42\nR R1 21\nG T L T1 T2 N1\nG T L T1 T3 N2\nG T L T2 T4 N2\nG T L T3 T4 N1\nG F L F1 F2 N1\nG F L F3 F4 N1\nG F L F1 F3 N2\nG F L F2 F4 N2\nG S C S1 R1\n##Output:\n

Answer:

S 1 S B 38 D 1\nP 1\nL +\nO S1 R1\nX N -N1 P 1

Example 5

Instruction:

Generate the CAD sequence based on the input SVG sequence.##Input:\nP F1 5 29\nP F2 11 29\nP F3 11 32\nP F4 16 32\nP F5 21 32\nP F6 46 32\nP F7 47 32\nP F8 57 32\nP F9 5 34\nP F10 11 34\nP F11 16 34\nP F12 21 34\nP F13 46 34\nP F14 47 34\nP F15 57 34\nP S1 27 29\nP S2 32 29\nP S3 37 29\nP S4 40 29\nP S5 32 31\nP S6 37 31\nP S7 40 31\nP S8 27 34\nP S9 32 34\nP T1 27 15\nP T2 57 23\nP T3 10 25\nP T4 4 25\nP T5 57 28\nP T6 16 28\nP T7 47 28\nP T8 8 29\nP T9 21 30\nP T10 46 30\nP T11 8 31\nP T12 1 31\nP T13 58 32\nP T14 57 34\nP T15 57 35\nP T16 16 36\nP T17 49 36\nP T18 8 37\nP T19 11 40\nP T20 54 40\nP T21 2 44\nP T22 63 44\nP T23 62 27\nP T24 7 24\nP T25 50 25\nP T26 46 29\nN N1 1\nN N2 4\nN N3 4\nN N4 5\nN N5 5\nN N6 6\nN N7 7\nN N8 8\nN N9 9\nN N10 9\nN N11 12\nN N12 12\nN N13 13\nN N14 13\nN N15 14\nN N16 16\nN N17 17\nN N18 23\nN N19 25\nN N20 43\nN N21 46\nN N22 51\nN N23 63\nR R1 2\nR R2 3\nR R3 5\nR R4 7\nR R5 12\nG T L T1 T7 N18\nG T L T3 T9 N11\nG T L T6 T1 N17\nG T L T6 T9 N4\nG T L T7 T15 N12\nG T L T9 T10 N19\nG T L T12 T4 N7\nG T L T12 T19 N14\nG T L T15 T22 N16\nG T L T18 T6 N13\nG T L T18 T19 N3\nG T L T20 T15 N6\nG T L T19 T20 N20\nG T L T20 T13 N9\nG T L T21 T18 N10\nG T L T21 T22 N23\nG T C T5 R2\nG T C T8 R2\nG T C T16 R1\nG T C T17 R1\nG T A T2 T23 T13 C T5 R R3\nG T A T4 T24 T3 C T11 R R4\nG T A T10 T25 T2 C T14 R R5\nG T A T10 T26 T7 C T14 R R5\nG F L F1 F9 N2\nG F L F1 F2 N5\nG F L F2 F10 N2\nG F L F3 F8 N21\nG F L F4 F11 N1\nG F L F5 F12 N1\nG F L F6 F13 N1\nG F L F7 F14 N1\nG F L F8 F15 N1\nG F L F9 F15 N22\nG S L S1 S8 N2\nG S L S2 S9 N2\nG S L S3 S6 N1\nG S L S4 S4 N15\nG S L S4 S7 N1\nG S L S5 S7 N8\nG S L S8 S9 N5\n##Output:\n

Answer:

S 1 T B 29 D 1\nP 1\nL +\nO T8 R2\nL +\nO T5 R2\nL +\nO T16 R1\nL +\nO T17 R1\nL +\nE T12 T19 N14\nE T19 T20 N20\nE T20 T13 N9\nA - T13 T2 C T5 R R3\nA + T10 T2 C T14 R R5\nE T10 T9 N19\nE T9 T3 N11\nA - T3 T4 C T11 R R4\nE T12 T4 N7\nS 2 T B 34 S 1\nP 2\nL +\nO T17 R1\nP 3\nL +\nE T1 T7 N18\nA + T10 T7 C T14 R R5\nE T10 T9 N19\nE T9 T6 N4\nE T1 T6 N17\nP 4\nL +\nO T16 R1\nP 5\nL +\nE T21 T22 N23\nE T22 T15 N16\nE T20 T15 N6\nE T19 T20 N20\nE T19 T18 N3\nE T21 T18 N10\nP 6\nL +\nE T15 T7 N12\nA + T10 T7 C T14 R R5\nE T10 T9 N19\nE T9 T6 N4\nE T6 T18 N13\nE T19 T18 N3\nE T19 T20 N20\nE T20 T15 N6\nL +\nO T16 R1\nL +\nO T17 R1\nX N N2 P 1\nX C -N1 P 4 2 5 3 6

Figure A1: Five instruction and answer examples used in our main paper.

Method	Token Type	Entropy ↓	Top-1 Prob. ↑	Target Prob. ↑
w/ Pointer	All pointers	0.143	0.942	0.923
	Point pointers ($\mathcal{P}$)	0.222	0.911	0.881
	Length pointers ($\mathcal{N}$)	0.008	0.997	0.995
	Radius pointers ($\mathcal{R}$)	0.047	0.978	0.969
w/o Pointer	All corresponding tokens	0.384	0.713	0.641
	Point values	0.421	0.624	0.552
	Length values	0.097	0.802	0.748
	Radius values	0.406	0.771	0.719

Table 7: Next-token uncertainty under teacher forcing on test samples. The lower entropy and higher probabilities of the pointer-routing formulation illustrate the benefit of representing CAD construction as symbolic routing.

next-token distribution behaves like a pointer-like routing mechanism over symbolic entities defined in the input.

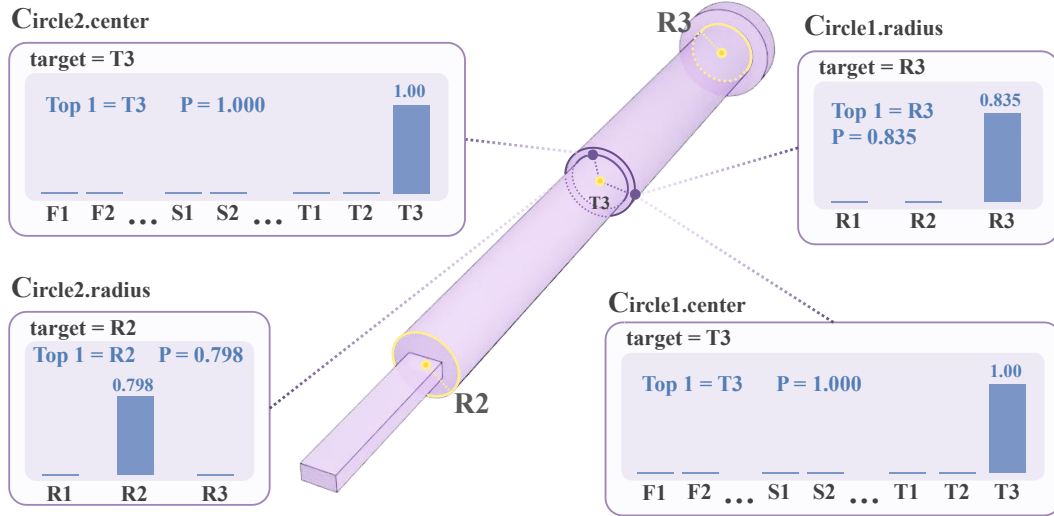


Figure A2: Qualitative visualization of the learned pointer-style routing behavior. The CAD object contains two highlighted circle entities. For each circle-related output slot, we show the model’s top-1 prediction, its probability, and the probabilities assigned to alternative candidate tokens. The correct symbolic reference receives the dominant probability mass, indicating that the model performs slot-conditioned selection over reusable CAD entities rather than unconstrained token generation.

F Equivariance under Symbolic Renaming

To test whether the model truly performs symbolic routing rather than memorizing fixed identifier patterns, we conduct a symbolic renaming experiment. For each test sample, we randomly permute the symbolic names in the input sequence, including point, length, and radius identifiers, while keeping the underlying geometric structure unchanged. We then regenerate the CAD sequence from the renamed input and compare it against the expected renamed target sequence.

Table 8 reports the results test samples. The most important finding is that the generated outputs remain entirely within the renamed symbol table: the slot validity is 1.000 and the invalid reference rate is 0.000 for all token types. This indicates that the model consistently routes to valid renamed symbols instead of hallucinating unseen identifiers. Although the strict slot-wise exact match rate is lower, especially for point pointers, this metric is conservative because it requires exact agreement at the same output position and does not account for equivalent alternative orderings or symmetric choices. Overall, the results support the claim that the model is equivariant to symbolic renaming and that its predictions are governed by symbolic routing over the input-defined namespace.

Token Type	Slot Exact Rate $\uparrow$	Slot Validity $\uparrow$	Invalid Ref. Rate $\downarrow$
All pointer slots	0.961	1.000	0.000
Point pointers	0.908	1.000	0.000
Length pointers	0.976	1.000	0.000
Radius pointers	0.978	1.000	0.000

Table 8: Permutation equivariance under symbolic renaming on test samples. For each sample, the input point, length, and radius identifiers are randomly renamed, and the model is required to generate a correspondingly renamed CAD sequence. The results show that the model always stays within the renamed symbol table, yielding perfect slot validity and zero invalid references.

G Additional Qualitative Results

To further demonstrate the robustness and generalization capabilities of **ForgeCAD**, we present additional qualitative results in Fig. A3. These examples encompass a diverse array of industrial CAD models with varying levels of topological complexity, ranging from simple mechanical parts to highly detailed assemblies.

As observed, **ForgeCAD** consistently translates 2D orthographic projections into highly accurate 3D boundary representations (B-Reps). The model excels at preserving intricate geometric details, such as precisely aligned through-holes, complex multi-level extrusions, and symmetric features, which frequently confound traditional numerical regression baselines. By leveraging the proposed pointer-centric symbolic representation, our approach ensures that the strict spatial relationships and dimensional specifications defined in the input engineering drawings are reliably routed and rigorously executed in the final procedural generation. This further validates the efficacy of reformulating CAD generation as a discrete symbolic routing task.

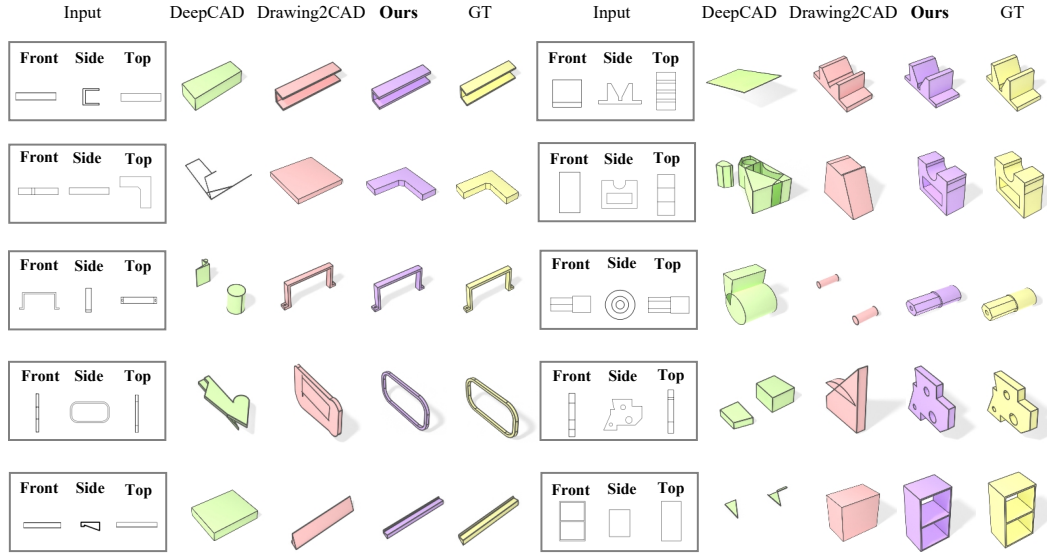


Figure A3: Additional qualitative results of **ForgeCAD** on the DeepCAD test set. The generated 3D CAD models exhibit high geometric fidelity and structural correctness, successfully capturing complex topologies, intricate internal features, and precise spatial arrangements specified by the 2D orthographic engineering drawings.

H Failure Cases, Limitations and Future Work

As illustrated in Fig. A4, we identify two primary categories of failure cases in **ForgeCAD**.

Incomplete View Representation. Following the established setting of prior work, our input engineering drawings relies on three orthographic projections with hidden-line removal. However, this formulation is not mathematically complete and inherently fails to capture features invisible from these standard views (e.g., internal cavities or blind holes on a bottom surface). Without a complete geometric representation, the model cannot reconstruct these occluded structures. Future work could resolve this bottleneck by incorporating dashed hidden lines or auxiliary views into the SVG sequence.

Routing Errors in Complex Assemblies. Even when the input views provide complete information, dense or stacked objects can occasionally cause symbolic routing errors. A typical manifestation is the model referencing an incorrect length scalar during an Extrude command. This indicates that visually entangled primitives with similar numerical values can still confound the pointer network. Future iterations may mitigate this by integrating explicit spatial reasoning objectives or enforcing stricter contextual constraints during token retrieval.

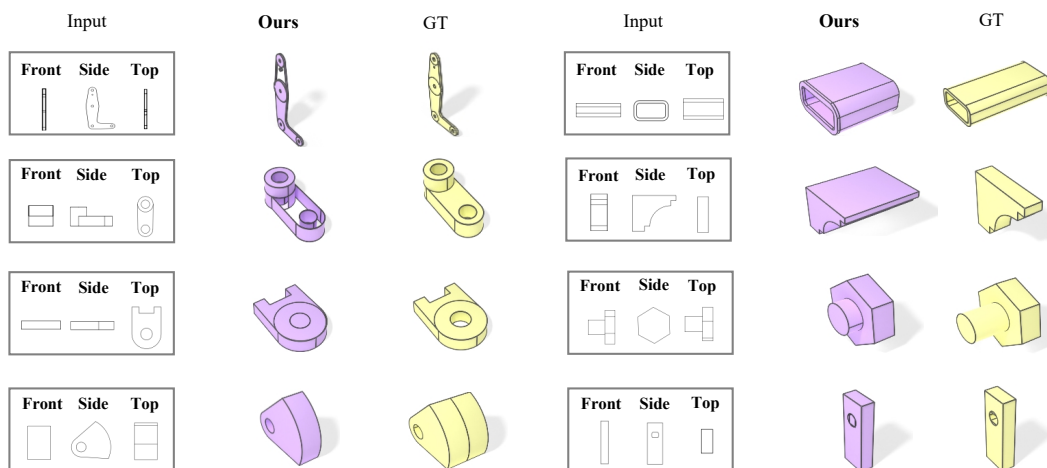


Figure A4: Typical failure cases of **ForgeCAD**.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [\[Yes\]](#)

Justification: The main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: We mention our limitations in Appendix H.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate “Limitations” section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren’t acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[N/A\]](#)

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: We disclose all information needed for reproduction in Section 4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will provide data, code, and instructions for the final paper.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://neurips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://neurips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We specifies all the training and test details in Section 4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report consistent performance across multiple runs and use fixed random seed settings to support the statistical significance of our results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We provide information about the computer resources used in Section 4.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: We have read and understood the code of ethics and have made every effort to adhere to it.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: Our work contributes to Drawings-to-CAD Generation. It does not impact society at large.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper poses no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All existing assets, including code, data, and models, are properly cited in the paper, and their licenses and usage terms are respected in accordance with the original sources.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset’s creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: We will release our codebase along with detailed README files to ensure usability and reproducibility in an anonymous manner.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [\[N/A\]](#)

Justification: This work does not use crowdsourcing or human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [\[N/A\]](#)

Justification: This work does not use crowdsourcing or human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.

- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [\[Yes\]](#)

Justification: We describe the usage of LLMs in Section 3 and details in Section 4.1.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.