Bidirectional Query-Driven Generation of Parametric CAD Sketch

Supplementary Material

This supplementary document provides additional technical details, experimental results, and further discussions to complement the main paper. It is organized into four sections:

- **Section 6: Parametric CAD Sketch Representation.** This section introduces the parametric sketch representation adopted in our work, including the tokenization scheme, quantization process, parameterization of geometric primitives, and the design of special tokens.
- **Section 7: Additional Implementation Details.** We present deeper insights for the core components of CADSketcher, describe the training and evaluation setups for all baseline models, and provide complete definitions and formulations of the metrics used in our experiments.
- **Section 8: Additional Experimental Results.** This section reports supplementary experiments covering early-stage sketch expansion, cross-dataset generalization, CAD workflow integration, user study, and extended ablation analyses, further validating the robustness and applicability of our framework.
- **Section 9: Discussion.** We highlight representative challenging cases, analyze their underlying causes, and discuss additional limitations and potential future research directions.

6. Parametric CAD Sketch Representation

Overview. In CADSketcher, we represent each parametric CAD sketch using a standard sequential format. As illustrated in Fig. 8, a 2D sketch (top) is formulated as an ordered sequence of tokens (bottom), consisting of primitive types, sketch parameters, and start/end mark tokens. In contrast to primitive-level representations commonly used in prior seq2seq approaches [49], our formulation is more compact (i.e., with reduced padding and higher information density), allowing the sequence to explicitly record the modeling process while capturing both geometric structure and design intent. In the following, we detail the full tokenization pipeline and the parameterization scheme.

Primitive Representations. Table 4 summarizes the components of our parametric sketch representation, including primitive types and several special tokens. We focus on three widely used primitives (Line, Arc, and Circle) in real-world CAD sketching workflows. Each primitive is parameterized by 2D coordinates (x, y) (or a radius r for circle), as detailed below:

- **Line**: Start Point and End Point
- **Arc**: Start Point, Middle Point, and End Point
- **Circle**: Center Point and Radius

Table 4. Token definitions for the parametric CAD sketch representation, including primitive parameter tokens and special structural tokens used to mark sequence boundaries and primitive types.

Category	Type	Token Value	Description
Primitive Token	x	$[\![7, \ldots, 70]\!]$	X coordinate
	y	$[\![7, \ldots, 70]\!]$	Y coordinate
	r	$[\![7, \ldots, 70]\!]$	Circle radius
Special Token	PAD	0	Padding token
	SOS	1	Start of sequence
	EOS	2	End of sequence
	s_a	3	Start of Arc
	s_c	4	Start of Circle
	s_l	5	Start of Line

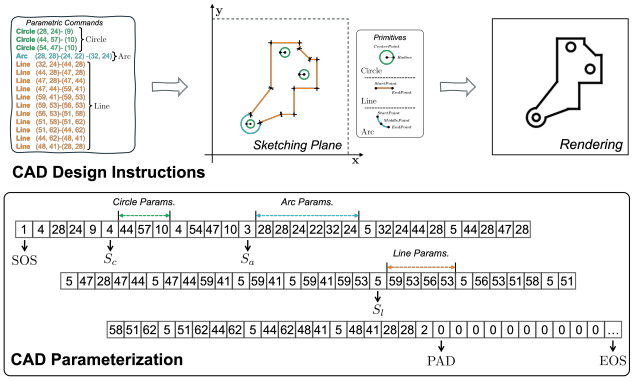

Figure 8. Overview of the parametric CAD sketch representation. A 2D sketch is expressed as an ordered instruction sequence composed of special boundary tokens, primitive type markers, and discretized geometric parameters. This sequential format preserves both geometric structure and modeling intent while maintaining a compact and unified tokenized representation.

Tokenization and Quantization. Following [42, 51], all geometric parameters are normalized and discretized via 6-bit quantization, enabling sketches to be represented as structured symbolic sequences and enhancing training stability. In addition, special tokens are employed to explicitly mark sequence boundaries and primitive types, while padding tokens are used to unify all sequences to a fixed length for efficient parallel training, as detailed in Table 4.

Implementation Details. We set the maximum sequence length to 130, which is sufficient to cover the vast majority of real-world CAD modeling scenarios. Following [41], each sketch contains at least 6 and at most 16 modeling primitives. We adopt the deduplication strategy of [42] to filter out redundant samples based on sequence similarity.

7. Additional Implementation Details

7.1. Implementation Details for CADSketcher

Besides the implementation details provided in Sec. 4.1 of the main paper, we present additional development insights and complementary implementation details for the core techniques used in this section.

Additional Details for Sketch Learning. In CADSketcher, we treat the input context as a semantic anchor to support stable bidirectional completion. Specifically, for an input partial sketch sequence $\mathcal{C}_{\text{par}}$, we first apply tokenization and hybrid positional encoding, and then feed the sequence into a sketch encoder $\mathcal{E}$ to obtain the partial sketch feature

$$f_{\text{par}} = \mathcal{E}(\tilde{\mathcal{C}}_{\text{par}}), \;\; \tilde{\mathcal{C}}_{\text{par}} = \text{Tokenize}(\mathcal{C}_{\text{par}}) + E_{\text{pos}}, \quad (8)$$

which captures the local topology, geometric layout, and underlying design intent of the current sketching state. These features serve as conditioning signals throughout decoding, ensuring that both prior and posterior queries remain aligned with the structural and semantic constraints imposed by the observed sketch.

To allow the model to fully leverage this semantic information during generation, we incorporate a cross-attention module inside the parameter decoder $\mathcal{D}$, enabling the bidirectional query vectors to explicitly attend to the partial sketch features. For the decoding state h_t at step t, the cross-attention process is defined as:

$$\text{Attn}(h_t, f_{\text{par}}) = \text{softmax}\left(\frac{(h_t W_Q)(f_{\text{par}} W_K)^\top}{\sqrt{d_k}}\right)(f_{\text{par}} W_V), \quad (9)$$

where W_Q, W_K, and W_V are learnable projection matrices, and d_k denotes the key (head) dimension. This mechanism enables each query stream to leverage not only its own decoding history but also the geometric structure encoded in the observed sketch, thereby preserving structural and semantic consistency during bidirectional completion. By continuously conditioning on f_{par} throughout decoding, CADSketcher avoids drifting away from the existing geometric structure and reliably infers the remaining modeling primitives.

Additional Details for Parallelism. For query-driven generative frameworks, the querying process is inherently sequential: departing from the standard next-token-prediction paradigm forces the model to decode tokens strictly step by step, thereby reducing learning efficiency. As described in Sec. 3.2 of the main paper, CADSketcher addresses this issue by structurally reorganizing the original modeling sequence and constructing two parallelized left/right streams that support *parallel context learning* for the prior and posterior sides. To prevent information leakage across invalid regions and avoid spurious dependencies, we design dedicated attention masks and padding masks for both branches, as illustrated in Fig. 9. Each query is constrained to attend only to the portion of the sequence that is valid for its position. For instance, the first query is allowed to access only $\mathcal{C}_{\text{par}}$, while the second query can additionally observe one further token, with subsequent queries progressively expanding their accessible context. The padding mask further ensures that any attention over invalid or out-of-range positions is strictly suppressed. This masking scheme eliminates cross-region information leakage during parallel modeling and allows both directional streams to be learned jointly within shared Transformer layers. During inference, however, since the model iteratively generates primitives on both sides, a standard causal mask is sufficient to support the generation process.

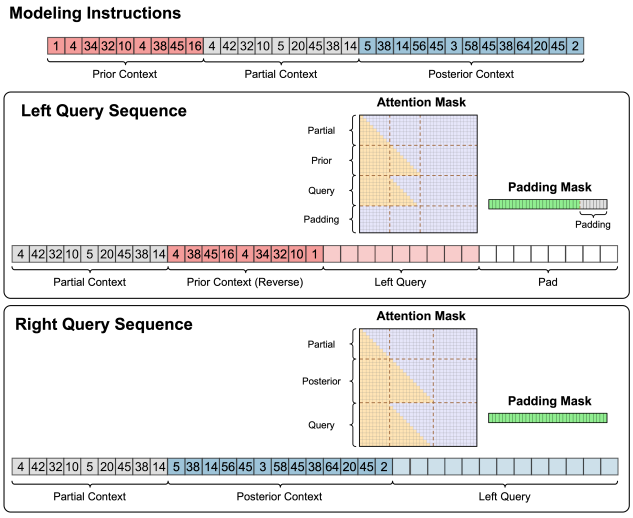

Figure 9. Illustration of the left and right query streams used for parallel context learning. The modeling instruction sequence is restructured into prior and posterior branches, each equipped with position-specific attention and padding masks that strictly confine each query to its valid context region, preventing cross-region information leakage while enabling parallel bidirectional modeling.

Additional Details for Confidence Gate. During bidirectional generation, determining whether to expand toward the prior (left) or posterior (right) side at each step is crucial for maintaining stable and coherent sketch completion. An inappropriate directional choice may lead to topological drift, accumulated errors, or unstable geometric expansion. To address this, CADSketcher employs a prediction-confidence–based direction selection strategy, referred to as the *Confidence Gate*. Given the current partial sketch state S_t, the model first predicts the type-probability distributions of the next primitive along both directions using their respective query vectors:

$$p_\theta^{\text{prior}} = p_\theta(c \mid S_t, \mathcal{Q}_{\text{prior}}), \;\; p_\theta^{\text{post}} = p_\theta(c \mid S_t, \mathcal{Q}_{\text{post}}). \quad (10)$$

To estimate the reliability of the two branches, we take the maximum predicted type probability from each distribution

as its confidence score:

$$\alpha_{\text{prior}} = \max\left(p_\theta^{\text{prior}}\right), \quad \alpha_{\text{post}} = \max\left(p_\theta^{\text{post}}\right). \tag{11}$$

These directional confidence scores are subsequently normalized via a temperature-controlled softmax to obtain the directional policy $\pi_\theta(d \mid S_t)$:

$$\pi_\theta(d \mid S_t) = \text{softmax}\left(\frac{1}{\kappa}\begin{bmatrix}\alpha_{\text{prior}} \\ \alpha_{\text{post}}\end{bmatrix}\right), \quad d \in \{\text{prior, post}\}, \tag{12}$$

where κ denotes the temperature coefficient that modulates the sharpness of direction selection. A smaller κ amplifies the difference between confidence scores and yields more deterministic expansion, whereas a larger κ encourages more balanced exploration. In the partial-to-full completion setting, we employ a deterministic policy by selecting the expansion direction via an argmax operation:

$$d_t = \arg\max_d \pi_\theta(d \mid S_t), \tag{13}$$

ensuring that decoding always proceeds along the more confident branch. Once the direction is determined, the model performs primitive-type and parameter prediction solely along that query branch, while the validity compiler further enforces geometric and structural constraints during primitive construction. This confidence-aware selection strategy avoids blind exploration in uncertain regions and significantly reduces topological drift, resulting in more coherent, stable, and executable sketch completion.

Additional Details for Validity Compiler. To ensure that generated modeling instructions satisfy the structural constraints of CAD sketches, we introduce a *Validity Compiler* (Alg. 1) during decoding to enforce parameterized structural rules. Each modeling primitive is represented as $c_t = (\tau_t, \phi_t)$, where τ_t denotes the primitive type command and ϕ_t its associated parameter set. Since different primitives follow distinct parameter structures, we define a *primitive schema* Γ that maps each primitive type to its corresponding parameter slots, where $\Gamma(\tau_t)$ denotes the set of valid slots associated with type τ_t.

During inference, the model first predicts the primitive type distribution conditioned on the current sketch state S_t and query direction $\mathcal{Q}_d$, selecting the most probable type τ_t in the *partial-to-full completion* setting (while temperature-controlled sampling is used in the *early-stage expansion*). The compiler then activates the parameter slots specified by $\Gamma(\tau_t)$ and restricts parameter decoding to these valid positions, masking out incompatible slots. The predicted type and parameters are finally assembled into the primitive $c_t = (\tau_t, \phi_t)$, ensuring that generated instructions conform to the structural rules of CAD primitives and remain geometrically executable.

Additional Details for Context Updating. During sketch completion, CADSketcher employs an *iterative context updating* mechanism to maintain a dynamic understanding of the evolving sketch state. At each step, the newly generated primitive is inserted into the corresponding side of the context (prior or posterior) rather than simply appended to the end of the sequence. A key detail is that the definition of the partial context varies depending on which direction will be expanded next. For clarity, consider a local sequence $\mathcal{C}_{\text{par}} = \{c_{t-1}, c_t, c_{t+1}\}$, where c_{t-1} and c_{t+1} are primitives newly generated on the left (prior) and right (posterior) sides, respectively, and c_t serves as the original partial context anchor:

- **For the next left expansion:** The model regards $\{c_t, c_{t+1}\}$ as the partial context for positional encoding and feature extraction, while c_{t-1} remains on the expand-side for the next generation step.
- **For the next right expansion:** The model treats $\{c_{t-1}, c_t\}$ as the partial context, and c_{t+1} is kept on the expandable side, serving as the region eligible for further generation.

The updated context sequence is then re-encoded with hybrid positional encoding and passed through the sketch encoder to obtain refreshed partial sketch features f_{par}. Meanwhile, the boundaries of the left and right query streams automatically realign with the new context, ensuring that subsequent inference steps remain synchronized with the current geometric structure. This iterative refreshing process effectively prevents semantic drift caused by static-context assumptions and enables the bidirectional completion procedure to follow the sketch evolution while maintaining geometric coherence and structural consistency.

Algorithm 1 Validity Compiler for Primitive Decoding

Require: Current sketch state S_t, direction query $\mathcal{Q}_d$, primitive schema Γ
Ensure: Generated primitive $c_t = (\tau_t, \phi_t)$
1: Predict primitive type distribution

$$p_\theta(\tau \mid S_t, \mathcal{Q}_d)$$

2: $\tau_t \leftarrow \arg\max_\tau p_\theta(\tau \mid S_t, \mathcal{Q}_d)$
3: **for all** $s \in \Gamma(\tau_t)$ **do** ▷ activate slots defined by primitive schema
4: Predict parameter tokens conditioned on $(S_t, \tau_t, \mathcal{Q}_d, s)$
5: **end for**
6: Assemble primitive $c_t = (\tau_t, \phi_t)$
7: **return** c_t

7.2. Implementation Details for Baseline Methods

This section provides detailed implementation setups for all baseline methods compared in the main paper, including Vitruvion [42], CAD-VLM [51], DeepCAD [49], as well as our tailored variants, Dual-AR Sketcher and Dec-Only Sketcher. For a fair comparison, all models are trained and evaluated following the same instruction representation, data split, and metric protocols described in Sec. 4 and 7.3.

Vitruvion. For Vitruvion [42], we adopt the publicly released implementation from its official repository. Since its original completion setting assumes a prefix-based partial input, we modify the data structure to support the arbitrary-span partial context required in our task. Specifically, for non-prefix contexts, the observed segment (partial context) is shifted to the beginning of the sequence, and the remaining primitives are appended in their original order. Apart from this, all hyperparameter configurations remain identical to those in the original implementation.

CAD-VLM. For CAD-VLM [51], it represents a cross-modal CAD sketch generation framework based on visual–symbolic fusion. Since the official implementation is not publicly available, we reproduce the model following the original paper. Specifically, we replicate the dual-stream encoder–decoder architecture of CAD-VLM, where the image branch adopts a ViT-MAE-Base pretrained model with an input resolution of 224×224, and the text branch uses a pretrained CodeT5+ model. Visual and textual embeddings are linearly projected into a shared latent space and concatenated, followed by cross-modal alignment via image–text contrastive learning to obtain unified multimodal representations. The model is trained using the AdamW optimizer with an initial learning rate of 3×10^{-4}, a batch size of 32, and for 30 epochs. The overall training objective combines image–text contrastive loss, language modeling loss, and image reconstruction loss. During inference, we adopt the same data preprocessing strategy as Vitruvion to support arbitrary-span completion. The partial context sequence and its corresponding rendered image are provided as inputs to the CodeT5+ text decoder, which autoregressively generates the complete parametric sketch instruction sequence. Further implementation details can be found in their original paper.

DeepCAD. For DeepCAD [49], we adopt its official architecture with consistent hyperparameter settings as reported in the original implementation. Since the model was designed for full CAD modeling, we adapt it to focus solely on parametric sketch generation. The model is trained for 200 epochs using a token-level cross-entropy loss. During completion inference, arbitrary-span partial contexts are encoded into a latent representation, from which the decoder reconstructs the complete sketch modeling sequence.

Dual-AR Sketcher. We implement Dual-AR Sketcher as a dual-branch autoregressive framework. The architecture shares the same sketch encoder and parameter decoder configuration as CADSketcher but maintains two independent parameter decoders, denoted as left and right branches, to separately model the prior and posterior directions. During both training and inference, Dual-AR Sketcher takes the partial context as input, and each branch follows the standard next-token prediction paradigm. The left branch sequence (prior context) is further reversed at the primitive level. All other hyperparameter settings are kept consistent with those of CADSketcher. This model can be directly applied to the arbitrary-span partial sketch completion task and serves as a dual-branch extension of the standard autoregressive architecture.

Dec-Only Sketcher. Inspired by Vitruvion, we implement GPT-style decoder-only architecture. The model removes the sketch encoder and employs two independent parameter decoders for the prior and posterior branches, each taking the partial context as input to autoregressively generate the completion sequence. Dec-Only Sketcher follows the same training, inference, data, and hyperparameter settings as Dual-AR Sketcher. This model is designed to assess the effectiveness of a purely decoder-based architecture that adapts to arbitrary-span sketch completion without additional feature interaction.

7.3. Additional Details for Evaluation Metrics

In this section, we provide comprehensive definitions and implementation details of all evaluation metrics used in the main paper. All metrics are computed following the experimental protocols described in Sec. 4, ensuring consistency and fairness across different models and evaluation settings.

Partial-to-full Completion Evaluation. In this setting, the main goal is to complete the faithful remaining modeling instructions given a partial sketch context. Following [51], we employ a set of multi-level structural and geometric consistency metrics to comprehensively evaluate the correctness and executability of the generated results. Specifically, the evaluation includes *Sketch Accuracy* (ACC_{skt}), *Primitive Accuracy* (ACC_{pri}), and *Token Accuracy* (ACC_{tok}). Among them, ACC_{skt} measures whether the generated sketch exactly matches the ground-truth sketch; ACC_{pri} indicates whether the generated sketch contains at least one correct primitive corresponding to the ground truth; and ACC_{tok} quantifies the average token-level prediction accuracy. The three metrics are formally defined as:

$$\text{ACC}_{\text{skt}} = \frac{N_s}{N}, \quad \text{ACC}_{\text{pri}} = \frac{N_e}{N}, \quad \text{ACC}_{\text{tok}} = \frac{1}{N}\sum_{i=1}^{N} \frac{n_i^{\text{c}}}{n_i^{\text{t}}}, \tag{14}$$

where N is the total number of sketches in the test set, N_s denotes the number of fully correct sketches, N_e counts the number of sketches that contain at least one correctly generated primitive, and n_i^{c} and n_i^{t} denote the number of correctly predicted tokens and the total number of tokens in the i-th ground-truth sketch, respectively.

In addition, we compute the *F1-score* (F1) and *Invalid Rate* (IR) to measure overall executability and stability. The F1 score is defined as the harmonic mean of precision and recall for each sketch, reflecting the completeness and redundancy of the generated primitives, while IR measures the ratio of sketches that fail during execution in the On-

shape CAD geometric kernel. The definitions are given by:

$$\text{F1} = \frac{1}{N}\sum_{i=1}^{N}\frac{2\times P_i \times R_i}{P_i + R_i}, \quad \text{IR} = \frac{N_{\text{inv}}}{N}, \tag{15}$$

where $P_i = \frac{n_i^{\text{cp}}}{n_i^{\text{p}}}$ and $R_i = \frac{n_i^{\text{cp}}}{n_i^{\text{gt}}}$ denote the precision and recall for the i-th sketch, respectively. Here n_i^{cp} represents the number of correctly predicted primitives, n_i^{p} and n_i^{gt} denote the numbers of predicted and ground-truth primitives in the i-th sketch, respectively, and N_{inv} is the number of invalid completions that fail during geometric solving.

Early-stage Expansion Evaluation. In this scenario, the model starts from minimal initial geometry and incrementally generates a complete sketch structure. To evaluate its performance under this highly unconstrained and diverse setting, we follow [49, 53] and measure generation fidelity, validity, and diversity using five complementary metrics: *Coverage* (COV), *Minimum Matching Distance* (MMD), *Jensen-Shannon Divergence* (JSD), *Unique Score* (Unique), and IR (consistent with previous sections).

COV evaluates the diversity of generated samples by computing the fraction of reference sketches $\mathcal{X}$ that are matched by at least one generated sample in $\mathcal{G}$. For each generated sample $g_j \in \mathcal{G}$, the reference sample $x_i \in \mathcal{X}$ with the smallest Chamfer distance is identified. The set of such nearest reference samples is then used to measure how many distinct reference sketches are covered by the generated set. Formally,

$$\text{COV}(\mathcal{X}, \mathcal{G}) = \frac{\left|\{\arg\min_{x_i\in\mathcal{X}} d_{\text{CD}}(x_i, g_j) \mid g_j \in \mathcal{G}\}\right|}{|\mathcal{X}|}, \tag{16}$$

where $d_{\text{CD}}(x_i, g_j)$ denotes the Chamfer distance between x_i and g_j. A higher COV value indicates that generated sketches cover a larger portion of geometric modes in the reference set, reflecting better diversity of the generated distribution.

MMD measures the geometric fidelity of the generated distribution. For each reference sample $x_i \in \mathcal{X}$, the Chamfer distance to its nearest generated sample $g_j \in \mathcal{G}$ is computed, and MMD is defined as the average of these minimal distances:

$$\text{MMD}(\mathcal{X}, \mathcal{G}) = \frac{1}{|\mathcal{X}|}\sum_{x_i\in\mathcal{X}}\min_{g_j\in\mathcal{G}} d_{\text{CD}}(x_i, g_j), \tag{17}$$

and lower MMD values indicate that the generated sketches are geometrically closer to the real distribution, reflecting high-fidelity reconstruction.

JSD evaluates the global distributional similarity between the generated and reference samples. All sketches are discretized into grid-based occupancy maps, from which the average occupancy distributions $P_{\mathcal{G}}$ and $P_{\mathcal{X}}$ are computed. The divergence is then defined as

$$\text{JSD}(P_{\mathcal{X}}, P_{\mathcal{G}}) = \frac{1}{2}\text{KL}(P_{\mathcal{X}} \parallel M) + \frac{1}{2}\text{KL}(P_{\mathcal{G}} \parallel M), \tag{18}$$

where $M = \frac{1}{2}(P_{\mathcal{X}} + P_{\mathcal{G}})$, KL denotes the Kullback–Leibler divergence. Smaller JSD values indicate that the generated and reference samples share more consistent spatial distributions, reflecting stronger global alignment.

Unique measures the creativity of the generated instruction sequences relative to the training data. All modeling instruction sequences are quantized into discrete token representations, and the proportion of distinct sequences within the generated set (i.e., sequences that do not duplicate any other generated sample) is computed as

$$\text{Unique} = \frac{|\text{unique sequences}|}{|\text{generated sequences}|}. \tag{19}$$

A higher Unique score indicates that the model produces a larger variety of non-repetitive geometric patterns, reflecting stronger generative diversity and better out-of-distribution exploratory capability.

Considering the large scale of our evaluation set, we randomly sample 1,000 sketches from the reference dataset and generate 3,000 samples using our model to compute all metric scores, following the protocol in [49, 53]. To ensure statistical reliability, the evaluation process is repeated three times, and the final results are reported as the averaged scores across runs.

8. Additional Experiments

To further substantiate the effectiveness and robustness of our framework, we present a series of supplementary experiments that extend beyond the results discussed in the main paper. These evaluations provide a broader examination of the model under diverse sketching conditions, data distributions, and downstream application scenarios.

8.1. Additional Experimental Results

This section reports additional results on early-stage sketch expansion, cross-dataset generalization, user study, and CAD workflow integration, offering complementary evidence for the stability, generalization capacity, and practical applicability of CADSketcher.

Additional Early-stage Sketch Expansion Results. Fig. 10 presents additional qualitative comparisons for the early-stage sketch expansion task. Starting from highly coarse partial inputs containing only a small subset of primitives, all methods are required to expand the sketch under a stochastic sampling setup (temperature $= 0.9$). Across a wide range of examples, CADSketcher consistently produces geometrically coherent and structurally plausible expansions while preserving key relationships such as symme-

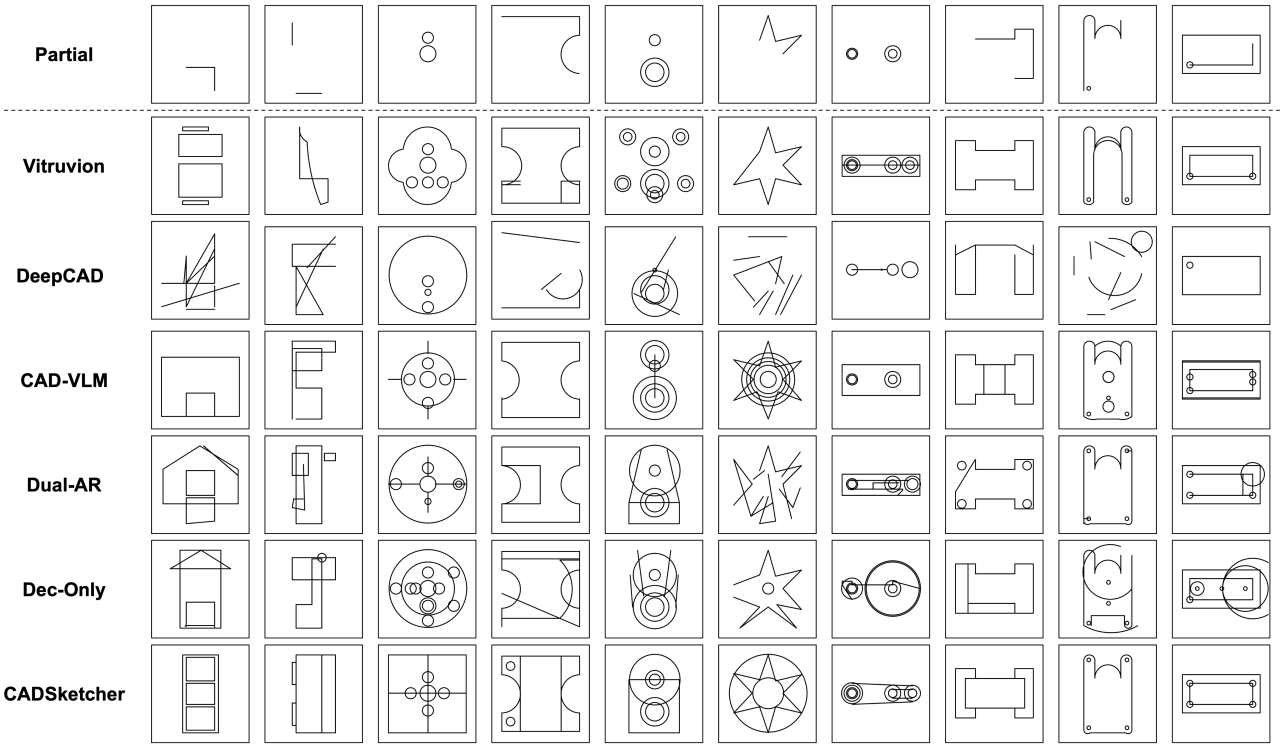

Figure 10. Qualitative comparison for the early-stage sketch expansion task on the SketchGraphs dataset [41]. Each example shows the partial input followed by expansion results from baseline methods and CADSketcher.

Table 5. Cross-dataset quantitative results for the early-stage sketch expansion task on the CAD as a Language dataset [10].

Methods	COV	MMD	JSD	Unique	IR
Vitruvion [42]	69.3	1.51	2.85	93.3	1.12
DeepCAD [49]	62.2	2.66	4.22	85.3	39.9
CAD-VLM [51]	71.3	1.43	1.67	93.8	4.26
Dual-AR	72.0	1.56	1.91	94.9	3.93
Dec-Only	57.6	3.28	3.35	99.6	3.37
CADSketcher	75.2	1.37	0.73	94.2	0.52

try, parallelism, and alignment. By contrast, baseline methods frequently exhibit over-expansion (e.g., Dec-Only), disrupted geometric relations (e.g., DeepCAD), or incomplete and unclosed sketch structures (e.g., Dual-AR). These supplementary results further corroborate the robustness of CADSketcher in conceptual drafting scenarios and highlight its capability for stable geometric reasoning and creative yet well-formed design exploration.

Additional Cross-Dataset Evaluation Results. To further assess the generalization capability of our method under distribution shift, we present additional cross-dataset results in this section. Fig. 11 provides qualitative visualizations for the partial-to-full sketch completion task on the CAD as a Language dataset [10]. Even without any dataset-specific fine-tuning, CADSketcher consistently produces completions that are structurally complete, geometrically precise, and semantically aligned with the underlying design intent. In contrast, most baselines exhibit noticeable instability when applied to this out-of-distribution setting, frequently generating incomplete, redundant, or geometrically distorted primitives that disrupt global sketch consistency.

Besides, we further evaluate CADSketcher's cross-dataset performance on the early-stage sketch expansion task. Fig. 12 illustrates multiple rounds of expansion produced by our model (temperature $= 0.9$, consistent with the main paper) from identical coarse partial inputs containing only 20–50% of the original primitives. Across all cases, CADSketcher maintains strong geometric stability while producing diverse structural variations, showcasing reliable shape reasoning and the ability to explore plausible conceptual designs. Quantitative results in Table 5 further support these findings: our method achieves leading performance across most metrics and exhibits the lowest invalid rate (IR), indicating that it not only generalizes well under distribution shift but also remains robust in generating valid and structurally coherent sketches.

Additional Integration and Application Results. To further demonstrate the practical applicability of our framework in real CAD workflows, Fig. 13 presents additional

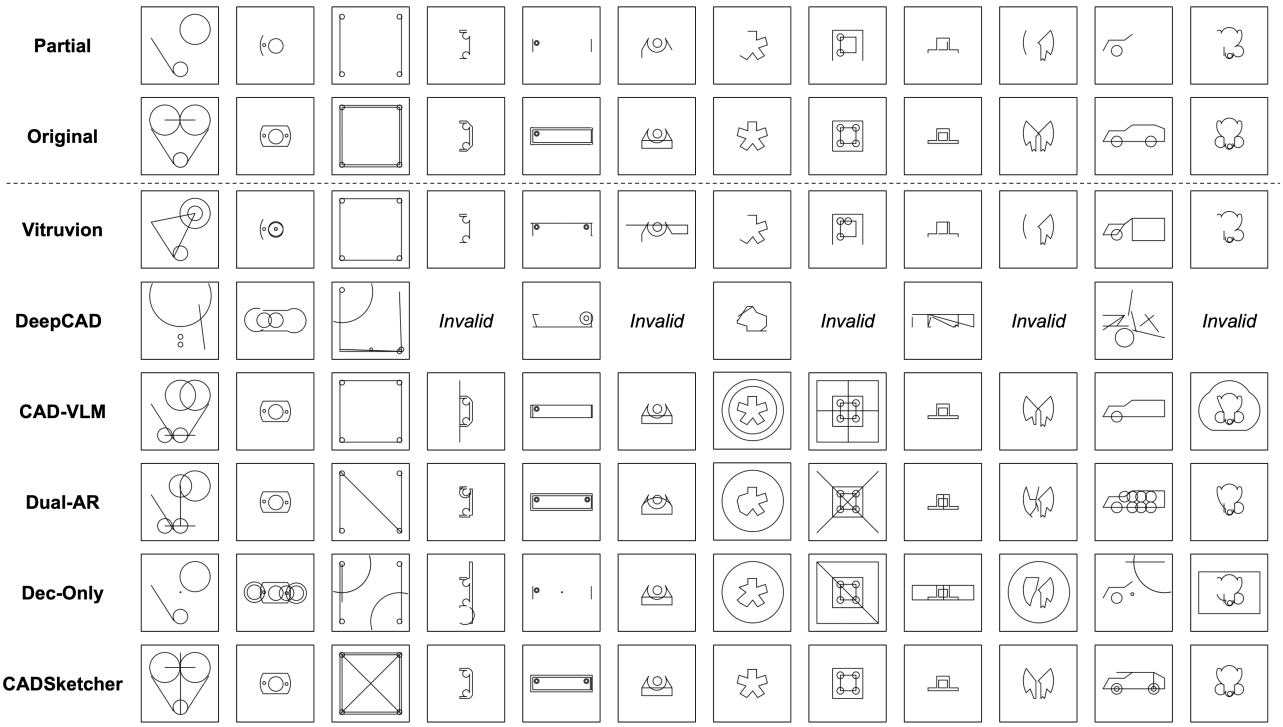

Figure 11. Cross-dataset qualitative comparison for the partial-to-full sketch completion task on the CAD as a Language dataset [10]. Each column shows the partial input, its reference sketch, and the corresponding completions from different methods. CADSketcher produces structurally coherent and geometrically well-aligned results, whereas baseline models often introduce redundant primitives or violate key geometric relationships.

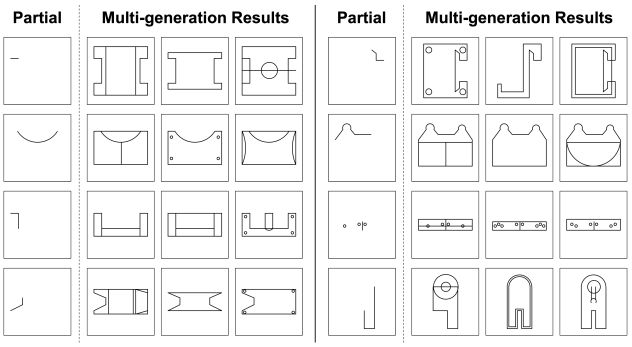

Figure 12. Cross-dataset qualitative results for the early-stage sketch expansion task on the CAD as a Language dataset [10].

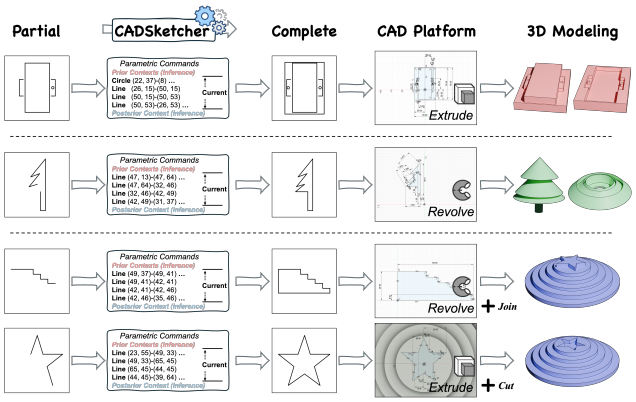

Figure 13. Additional examples of completing partial sketches and converting them into solid models within the standard CAD platform. Each single completed sketch supports multiple construction pathways, enabling the generation of diverse and valid 3D shapes.

downstream integration examples. By converting the generated completion sketches into executable parametric programs via CAD API tools, all outputs can be reliably processed by the geometric solver and used to construct topologically valid solid models within the commercial CAD platform. Notably, a single completed sketch can yield multiple distinct 3D designs depending on the construction operations applied (e.g., *extrude*, *revolve*, or combinations), highlighting the flexibility of CADSketcher in real design scenarios. These supplementary results confirm that the generated sketches are not only structurally complete and geometrically coherent, but also directly usable in downstream modeling pipelines, enabling seamless integration into standard CAD workflows.

Table 6. User study results evaluated by 23 CAD professionals on 20 sketch completion cases using a 10-point scale.

Methods	*Plausibility*	*Consistency*	*Editability*
CADSketcher	8.53	8.65	8.67

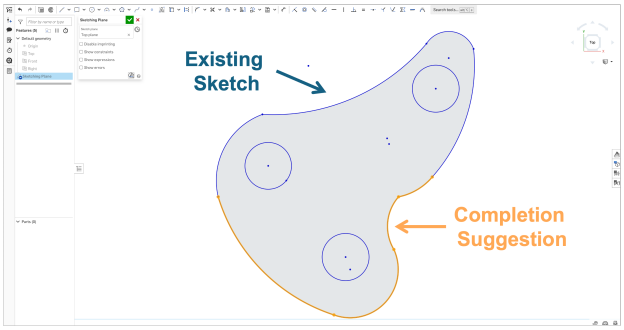

Figure 14. Interactive deployment prototype. Given a partial sketch (blue), the system generates completion suggestions (orange) that extend the geometry while preserving structural coherence.

User Study. To assess the effectiveness of CADSketcher in real design scenarios from a human perspective, we conduct a user study involving 23 professional participants with CAD engineering experience. We randomly select 20 sketch completion cases, where participants evaluate the generated results in terms of *Plausibility*, *Consistency*, and *Editability* using a 10-point scale (1–10). Specifically, *Plausibility* reflects the semantic rationality of the generated structure, *Consistency* measures the geometric coherence between the completed primitives and the existing sketch, and *Editability* evaluates the ease of subsequent parametric modification. As shown in Table 6, we achieve high average scores across all criteria, suggesting that the generated sketches are generally perceived as semantically reasonable, geometrically coherent, and convenient for subsequent editing.

Deployment and Application. To further validate the practical usability of the proposed method, we deploy the model as a lightweight interactive assistance module integrated into a standard CAD modeling workflow. In practice, users first create a partial sketch in the CAD platform as the initial geometric context. The current sketch state is then converted into a parametric instruction sequence and fed into the model for inference. As illustrated in Fig. 14, the system predicts plausible subsequent modeling primitives conditioned on the existing context, providing completion suggestions that extend the current geometry while maintaining structural consistency and executability.

Table 7. Ablation study of the hybrid positional encoding strategy on the partial-to-full completion task evaluated on the SketchGraphs dataset [41]. Relative-position encoding (pos.) is used as the baseline, and individual components are incrementally added.

Methods	ACC_{tok}	ACC_{pri}	ACC_{skt}	F1	IR
with pos.	70.7	74.3	42.6	57.0	0
with pos. + dir.	72.3	75.3	44.3	58.2	0
with pos. + ord.	71.2	74.1	42.5	57.6	0
with pos. + type	71.7	74.8	43.8	57.8	0
Full Model	72.8	75.6	45.6	59.2	0

Table 8. Directional stability analysis on the partial-to-full completion task evaluated on the SketchGraphs dataset [41].

Methods	ACC_{tok}	ACC_{pri}	ACC_{skt}	F1	IR
Fixed	68.8	68.3	37.8	51.5	0
Random	67.4	67.7	34.2	49.1	0
Heuristic	68.6	68.9	36.3	50.2	0
CADSketcher	72.8	75.6	45.6	59.2	0

8.2. Additional Ablation Study Results

Hybrid Positional Encoding Analysis. As a supplement to the ablation studies presented in the main paper, we conduct a finer-grained analysis of the hybrid positional encoding strategy. Since relative-position encoding (pos.) is indispensable for maintaining sequence interpretability during inference, we take it as the default baseline (i.e., the w/o HyPE setting in Table 3) and incrementally incorporate three additional components: a global directional encoding (with pos. + dir.), a local slot-ordering encoding (with pos. + ord.), and a local type encoding (with pos. + type). As reported in Table 7, each component contributes measurable performance improvements, while the invalid rate remains unaffected across variants. The full hybrid configuration, which jointly integrates both global and local positional cues, achieves the best overall performance, further validating its effectiveness in capturing the bidirectional and multi-scale semantic dependencies required for robust sketch completion.

Directional Stability Analysis. To analyze the impact of different direction-selection strategies during decoding, we compare our default confidence-gated policy with three alternatives: a fixed right-first strategy (Fixed), the random direction selection (Random), and a heuristic alternating strategy (Heuristic) that expands the sequence by alternately selecting the left and right directions. In all comparisons, only the direction-selection policy is changed and the other configurations remain identical. As shown in Table 8, CADSketcher maintains stable performance across all strategies, indicating consistent and reliable decoding behavior.

Table 9. Multi-run stability analysis on the partial-to-full completion task evaluated on the SketchGraphs dataset [41] (mean ± std over 5 runs with different random seeds).

Methods	ACC_{tok}	ACC_{pri}	ACC_{skt}	F1	IR
CADSketcher	72.90 ± 0.0038	75.81 ± 0.0025	45.82 ± 0.0020	59.41 ± 0.0016	0.0 ± 0.0

Multi-run Stability Analysis. We further evaluate the statistical stability of the proposed method through repeated runs. Under the same experimental settings, we perform five independent runs with different random seeds on the partial-to-full completion task and report the mean (± standard deviation) of the main metrics in Table 9. It can be seen that CADSketcher exhibits low variance across the token-, primitive- and sketch-level accuracies, as well as the F1 score, indicating stable overall performance.

9. Discussion

While CADSketcher performs strongly across diverse sketching scenarios, we still observe several challenging cases that highlight practical situations remaining difficult to handle. Representative examples are shown in Fig. 15 (all drawn from the *partial-to-full sketch completion* setting): although the model can produce topologically complete sketches, certain localized deviations may still appear. These challenging cases typically manifest as parameter (left), primitive (middle), or topology differences (right), each capturing subtle but meaningful discrepancies between the generated sketches and their original reference counterparts. From our experiments, these differences mainly stem from the fact that the completion task permits multiple plausible modeling trajectories. Since our goal is completion rather than exact reconstruction, the model is not constrained to replicate the original sketch verbatim, and such variations naturally emerge.

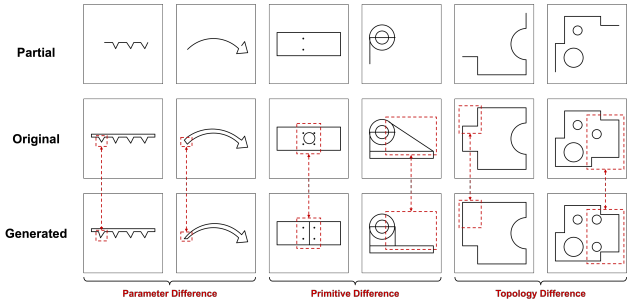

Figure 15. Representative challenging cases from the partial-to-full completion task, illustrating parameter, primitive, and topology differences between generated sketches and their reference counterparts (original).

Beyond above discussion and the data- and primitive-related limitations discussed in the Sec. 5 of the main paper, we identify two additional aspects that deserve further consideration. First, because the SketchGraphs dataset [41] is parsed directly from user-generated models on the *Onshape* platform, it inevitably contains noisy or semantically irrelevant samples (e.g., redundant primitives, dangling segments, or partially open sketches). Although we filter out entirely invalid cases (i.e., cannot form any meaningful sketches), residual noise remains and may still affect stable model learning. Second, CAD design is fundamentally a user-oriented activity, with individual modeling habits and stylistic preferences varying widely. Therefore, a promising direction lies in preference-aware sketching, where generative models adapt to designer-specific styles or task-dependent modeling behaviors, representing an important step toward more personalized and intelligent CAD systems.