DRAFTCODER:A BENCHMARK AND MODEL FOR DIMENSIONED MULTI-VIEW MECHANICAL DRAWING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mechanical engineering drawings convey part design intent through standardized views and complete dimensioning, serving as a key link between design and manufacturing. When the original 3D CAD models are unavailable, producing drawings from part images and key dimensions still relies on manual drafting. We introduce DraftCoder, which combines supervised fine-tuning and reinforcement learning with execution feedback to generate executable programs that produce editable multi-view DXF drawings with native dimension annotations. We design a Hierarchical Drawing-Consistency Reward (HDCR) that separately evaluates geometric measurements and dimension annotations, conditioning dimension and annotation rewards on view and geometric quality. We also construct DimDraft, a dataset and benchmark comprising 5,233 samples across nine categories of part structures, pairing image–text inputs with drawing programs and reference drawings. DraftCoder achieves a Scorable DXF Rate of 86.26%. With only 96 distinct training samples used during reinforcement learning, it improves Complete Annotation Match Rate and Layout Acceptance Rate by 10.29% and 9.12% relative to SFT, respectively. The dataset will be publicly released.

1 INTRODUCTION

Engineering drawings communicate the geometric structure and manufacturing requirements of parts through standardized views and dimension annotations, serving as an essential link between design, manufacturing, and inspection. When the original CAD model is unavailable and only part images and key dimensions are provided, producing engineering drawings still requires professional expertise and manual effort. Automatically generating fully dimensioned, editable engineering drawings therefore has practical value in reducing drafting costs and supporting subsequent design revisions and manufacturing collaboration.

Early methods enabled structured CAD generation by modeling sequences of sketch primitives, geometric constraints, and modeling operations (Para et al., 2021; Seff et al., 2022; Wu et al., 2021). Subsequent studies used pretrained models to translate point clouds or images into executable CAD programs, extending the capabilities of programmatic geometry construction and reconstruction (Rukhovich et al., 2025; Doris et al., 2025). However, these efforts remain primarily focused on constructing and recovering geometric design representations. Recent work has further enabled the generation of annotated DXF files (He et al., 2025) and the reconstruction of 3D CAD models using the views and dimensions in engineering drawings (Kim et al., 2026; Joglekar et al., 2026; Niu et al., 2025c). Nevertheless, annotation generation in the former mainly targets 2D sketches, while the latter relies on existing views and dimension information for reconstruction. Generating complete engineering drawings requires a model to infer spatial structure from observations of a part, derive the corresponding orthographic views, and organize dimensional requirements into annotations correctly associated with geometric entities, as illustrated in Figure 1. We therefore focus on coordinating these interdependent drafting requirements within a unified program generation framework to produce drawings with complete view geometry and consistent dimension annotations.

To this end, we propose **DraftCoder**, a multimodal program generation method for dimensioned multi-view mechanical drawings. Given a part's isometric image and a description of its key geometric dimensions, DraftCoder generates executable Python/ezdxf programs that jointly represent view geometry, spatial layout, and dimension annotations in editable DXF drawings. We first per-

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

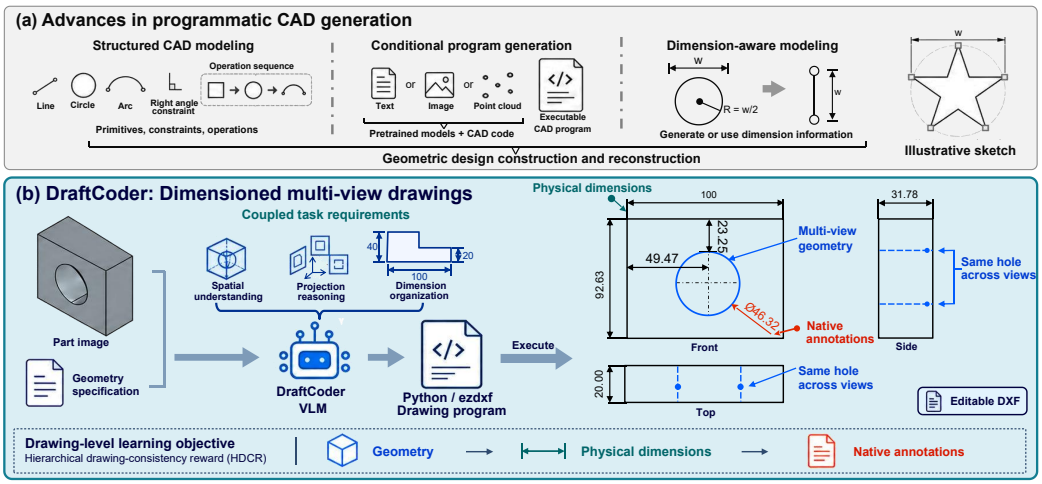


Figure 1: **Task positioning of DraftCoder.** (a) Existing research spans structured modeling, conditional program generation, and the generation and use of dimensional information, including annotated sketch generation and 3D reconstruction from existing views and dimensions. (b) Given a part image and geometric specifications, DraftCoder generates executable programs that jointly produce orthographic views and native dimension annotations associated with geometric entities, yielding editable multi-view DXF drawings.

form supervised fine-tuning to establish a mapping from image–text inputs to drawing programs. However, token-level supervision on reference programs does not directly optimize the quality of the drawings produced by execution. We therefore introduce reinforcement learning with execution feedback and design a Hierarchical Drawing-Consistency Reward (HDCR). This reward separately assesses geometric measurements and dimension annotations, using hierarchical aggregation to condition dimension and annotation feedback on foundational view and geometric quality, thereby guiding model optimization with the quality of the generated drawings.

To support training and systematic evaluation on this task, we construct **CODE**, a dataset of 5,233 samples spanning nine categories of part structures, pairing isometric images and geometric dimension descriptions with executable drawing programs and reference engineering drawings. By tracing construction information in the source CAD programs, we associate dimensional requirements with geometric entities in the projected views and generate drawings with visible contours, hidden lines, and native dimension annotations, together with their corresponding programs. The dataset covers profile dimensions, local feature positions, and thickness and depth dimensions, with annotations organized across multiple views to support learning the joint representation of geometry and dimensions. Building on this dataset, we establish a benchmark that evaluates the drawings produced by program execution in terms of views, geometry, dimensions, annotations, and layout, supporting comparisons across structural categories.

Our main contributions are as follows:

- **Joint generation of geometry and dimensions.** We introduce DraftCoder, which brings multi-view geometry and native dimension annotations into a unified program generation framework conditioned on a part's isometric image and key geometric specifications, producing editable engineering drawings.
- **Drawing quality as a hierarchical learning signal.** We develop HDCR to distinguish geometric measurement correctness from the values, types, and geometric bindings of native dimension annotations. Its hierarchical aggregation makes dimension and annotation rewards depend on foundational view and geometry quality, explicitly incorporating these dependencies into reinforcement learning.
- **Paired data and evaluation for multi-view drafting.** We establish CODE, linking part images and dimensional descriptions to executable programs and reference drawings across 5,233 samples and nine structural categories. Its coverage of visible and hidden geometry

and dimensions distributed across views supports systematic evaluation of geometry and dimension coordination.

2 Related Works

2.1 CAD and Drawing Program Generation

Learning-based CAD generation spans structured sketches and executable modeling programs. SketchGen (Para et al., 2021) and Vitruvion (Seff et al., 2022) model sketch primitives and geometric constraints, while CAD-Recode (Rukhovich et al., 2025) and CAD-Coder (Doris et al., 2025) generate CadQuery programs from point clouds and images, respectively. Methods for engineering drawings incorporate dimensional information: CAD-Coder (He et al., 2025) generates editable DXF drawings from text using `ezdxf`, whereas Drawing-Recode (Kim et al., 2026) and Ortho2CAD (Joglekar et al., 2026) reconstruct 3D CAD from annotated drawings. Beyond token-level supervision, CADFusion (Wang et al., 2025) incorporates visual feedback through preference optimization, while CAD-Coder (Guan et al., 2025) and CME-CAD (Niu et al., 2025c) apply reinforcement learning with geometric rewards to CAD code generation. We generate dimensioned multi-view mechanical drawing programs from a part image and geometric specifications. HDCR jointly evaluates executed geometry, physical dimensions, and native dimension annotations, conditioning annotation rewards on geometric and dimensional consistency.

2.2 CAD Datasets and Benchmarks

CAD datasets have expanded from construction histories to multimodal supervision and task-specific evaluation. Fusion 360 Gallery (Willis et al., 2021) and DeepCAD (Wu et al., 2021) provide parametric construction sequences. Text2CAD (Khan et al., 2024) adds textual annotations, while Omni-CAD (Xu et al., 2024) and CADFS (Pyatov et al., 2026) pair CAD representations with language and visual inputs. Drawing datasets provide richer annotation structure: ParaCAD (Niu et al., 2025a) includes geometric constraints and dimension associations, while TriView2CAD (Niu et al., 2025b) provides dimensioned three-view drawings and numerical reasoning tasks for bridge piers. Evaluation has also broadened: CADBench (Doris et al., 2026) assesses 3D CAD reconstruction across input modalities, BenchCAD (Zhang et al., 2026) covers generation, editing, and question answering, and CADTests (Mallis et al., 2026) checks geometric requirements using executable tests. We complement these resources with a dataset that pairs part images and geometric specifications with executable `ezdxf` programs. Its aligned multi-view geometry and dimension annotations support supervised learning and structured evaluation of mechanical drawing generation.

3 Method

3.1 Overview

As shown in Figure 2, DraftCoder takes a part image and textual geometric specifications as input and generates Python/ezdxf programs whose execution produces editable DXF drawings with orthographic views and native dimension annotations. Training comprises two stages: supervised fine-tuning learns the mapping from image–text inputs to drawing programs, and reinforcement learning further optimizes the policy using feedback on the resulting drawings.

3.2 Supervised Fine-Tuning for Engineering Drawing Programs

Given the paired dataset $\mathcal{D}_{\mathrm{SFT}} = \{(I_i, T_i, C_i^*)\}_{i=1}^{N}$, where I_i, T_i, and C_i^* denote the part image, textual geometric specifications, and reference code response, respectively, we update language-model adapters through LoRA while freezing the base model weights. We minimize autoregressive cross-entropy over target response tokens only:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\frac{1}{Z}\sum_{i=1}^{N}\sum_{t=1}^{n_i}\log \pi_\theta\big(c_{i,t}^* \mid I_i, T_i, c_{i,<t}^*\big), \qquad Z = \sum_{i=1}^{N} n_i, \tag{1}$$

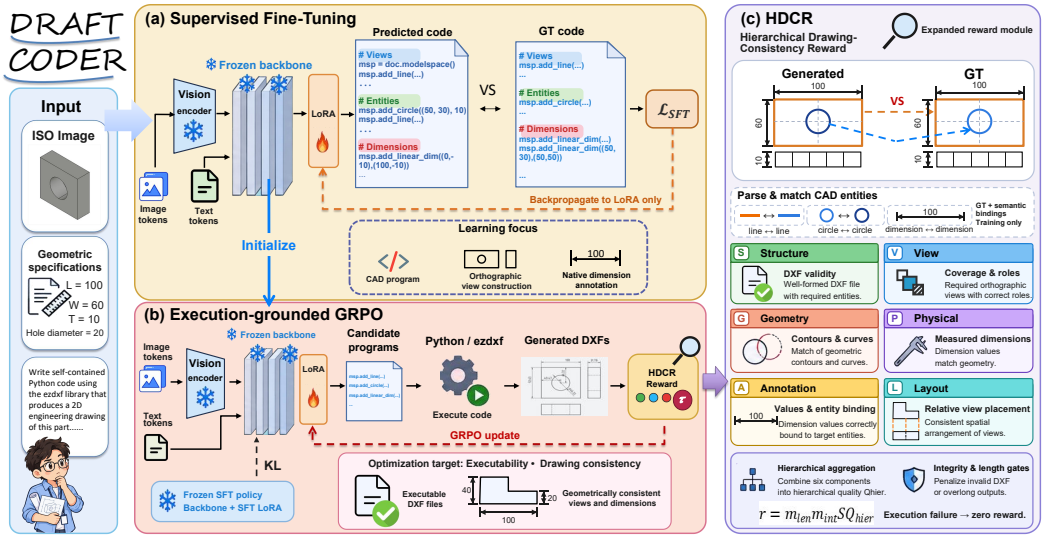


Figure 2: Overview of DraftCoder: supervised fine-tuning initializes drawing-program generation, followed by GRPO with HDCR feedback from the resulting DXF drawings.

where $c^*_{i,t}$ is token t of the reference response, n_i is its token count, and θ denotes the trainable adapter parameters.

3.3 Reinforcement Learning with Hierarchical Drawing-Consistency Reward

Different programs can produce equivalent drawings, while local parameter deviations can cause substantial geometric or annotation errors. To incorporate feedback on the resulting drawings, we initialize the policy from SFT, sample K candidate programs for each input $x = (I, T)$, execute and score them, and update the policy using GRPO.

Hierarchical drawing-consistency reward. HDCR compares generated and reference drawings along six dimensions: structural validity S, view coverage V, geometric consistency G, physical dimension consistency P, native annotation consistency A, and view layout consistency L. Here, $S \in \{0, 1\}$ is a validity gate, and the remaining components lie in $[0, 1]$. Using pre-established semantic bindings and a shared alignment that permits uniform scaling of the entire drawing, P directly compares geometric measurements without reading predicted annotation text. In contrast, A checks the presence, type, displayed value, and binding location of native `DIMENSION` entities, and penalizes redundant annotations. Training uses samples for which all six components are evaluable and the selected dimensional constraints have reliable semantic bindings.

Let τ be an auxiliary topological score combining primitive matching, contour closure, and penalties for extra fragments. To condition dimension and annotation rewards on foundational view and geometry quality, we define

$$H(a,b) = \begin{cases} \dfrac{2ab}{a+b}, & a+b>0, \\ 0, & a+b=0, \end{cases} \tag{2}$$

$$u = H(V,\tau), \qquad g = H(u,G), \qquad q = H(g,P),$$

The final reward is

$$r_{\mathrm{HDCR}} = m_{\mathrm{len}} m_{\mathrm{int}} S\left[0.35u(0.70 + 0.30L) + 0.30uG + 0.20gP + 0.15qA\right]. \tag{3}$$

Quality at each level contributes to the reward, while dimension and annotation contributions depend on upstream quality. The gate m_{len} assigns zero reward to truncated outputs, and m_{int} penalizes excess overlaps, duplicate entities, and degenerate entities relative to a reference-drawing baseline. Execution failures caused by candidate programs also receive zero reward.

Policy optimization. For each input, the candidate advantage is $\hat{A}_k = (r_k - \bar{r}(x))/(\sigma_B + \delta)$, where $\bar{r}(x)$ is the group mean reward, σ_B is the reward standard deviation in the current sampled generation batch, and δ is a numerical stability constant. We optimize a GRPO objective with token-level probability-ratio clipping and KL regularization relative to the frozen SFT policy, averaging the loss over valid tokens within each candidate. The full objective and notation are provided in Appendix A.

4 BENCKMARKING

4.1 DATASET CONSTRUCTION

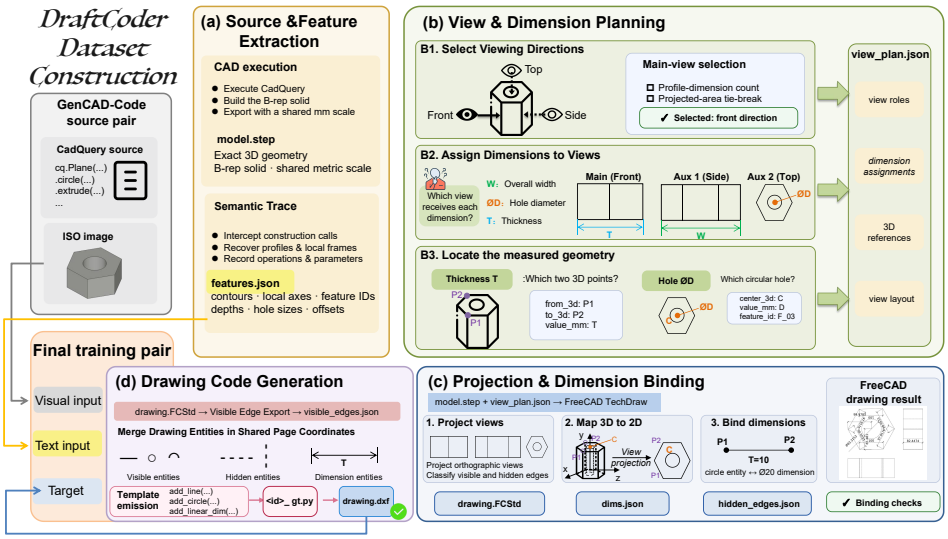


Figure 3: Overview of the dataset construction pipeline.

Source and Feature Extraction. Because a B-rep records final geometry and topology rather than construction history, it does not directly expose the dimensional basis of construction features. We therefore use GenCAD-Code's paired isometric images and CadQuery programs. Tracing sketches, extrusions, and Boolean operations yields contour geometry, local coordinate frames, feature locations, and construction parameters, which we store with dimension candidates in `features.json` for view planning and dimensioning (Appendix B.3).

View and Dimension Planning. From these features, we select the front-view direction by the number of profile dimensions associated with each sketch normal and determine the other orthographic views. Profile and hole dimensions are assigned to views whose viewing directions are normal to the corresponding sketch planes; extrusion thicknesses are assigned to views with viewing directions perpendicular to the extrusion direction, preserving true lengths. Using local frames and construction parameters, we express measurement references, including endpoints and circle centers, in a common 3D coordinate system. `view_plan.json` records these references, dimension types and values, and target views for projection and annotation binding (Appendix B.4).

Projection and Dimension Binding. Following the plan, FreeCAD projects the corresponding STEP model into orthographic views with visible and hidden edges. We project the 3D measurement references into their assigned views, bind editable dimension annotations to geometry, and produce annotated FCStd drawings. Dimension records and hidden lines are exported in a shared drawing coordinate system for program generation (Appendix B.5).

Drawing Program Generation and Sample Assembly. We combine visible geometry extracted from the FCStd drawings with the exported hidden lines and dimension records, and convert them into `ezdxf` drawing commands. Embedding geometry and annotation parameters in a standalone Python program enables DXF generation with editable dimension entities without FreeCAD or external intermediate files. Finally, templated part descriptions derived from structured features are

paired with the isometric images and programs, yielding image–text inputs with executable programs as supervised targets (Appendix B.6).

4.2 Dataset Composition and Analysis

Dataset Characteristics. The engineering drawings in CODE encode contours, visibility, and dimensional relationships (Table 1). Beyond outlines composed of straight and curved entities, nearly half of the drawings contain hidden lines, extending the target representation to occluded structures rather than visible boundaries alone. Dimensions specify not only linear distances, diameters, and radii, but also local feature positions and thickness or depth, linking geometry to its scale and spatial relationships. Moreover, 99.4% of samples with valid linked annotation records include dimension annotations in at least two views, making the organization of dimensions across views a common requirement. CODE therefore goes beyond tracing two-dimensional outlines: it requires determining structural visibility, expressing local spatial relationships, and coordinating geometry and dimensions across views. Definitions and computation of the annotation and visibility statistics are provided in Appendix B.2.

Category Structure. To distinguish the structural configurations encountered in engineering drawing generation, we organize samples into nine categories based on source-CAD profile types, inner loops or subtractive structures, and construction relationships. These categories are further organized into three construction-focused groups: *Basic profiles*, *Featured profiles*, and *Composite forms* (Figure 4). Basic profiles include polygonal, rectilinear, and circular contours; Featured profiles emphasize holes, cutouts, and curved contours; Composite forms distinguish combinations of polygonal, arc-bearing, and circular profiles. Figure 4(b) reveals two complementary structural changes. From Basic profiles to Featured profiles, the mean number of closed-profile inputs rises from 1.04 to 2.22, while the mean number of extrusion stages remains close to one, indicating richer profile composition within a construction stage. The Composite forms group averages 3.25 extrusion stages, additionally capturing multi-stage construction. This taxonomy thus captures differences in both internal profile structure and construction composition, enabling category-wise evaluation of how models handle distinct geometric configurations rather than merely different amounts of drawing content.

Table 1: Overview of the CODE construction collection.

Statistic	Value	Statistic	Value
Dataset scope		**Drawing geometry**	
Part samples	5,233	Geometric entities	304,953
Shape categories	9	Drawings with curves (%)	54.7
Train / validation / test	3,929 / 256 / 1,048	Drawings with hidden lines (%)	47.6
Part construction		**Dimension types**	
Extrusion-input profile stages	7,331	All native dimensions	32,925
Closed profile inputs	10,112	Linear dimensions	25,540
Polygonal / circular / arc-bearing loop inputs	3,833 / 4,796 / 1,483	Diameter dimensions	4,789
Parts with multi-loop inputs	1,602	Radius dimensions	2,596
Multi-stage boolean parts	953	**Verified bore geometry**	
Parts with union / cut	611 / 446	Verified cylindrical bore segments	2,535
Recorded bidirectional extrusions	547	Parts with verified bore segments	1,449
Part topology		**Spatial constraints**	
B-rep solids	5,171	Depth / thickness dimensions	6,410
B-rep faces	40,331	Local-position dimensions	5,221
B-rep edges	87,842	**View organization**	
B-rep vertices	58,379	Drawings dimensioned in at least two views (%)	99.4
Parts with cylindrical faces (%)	54.2		

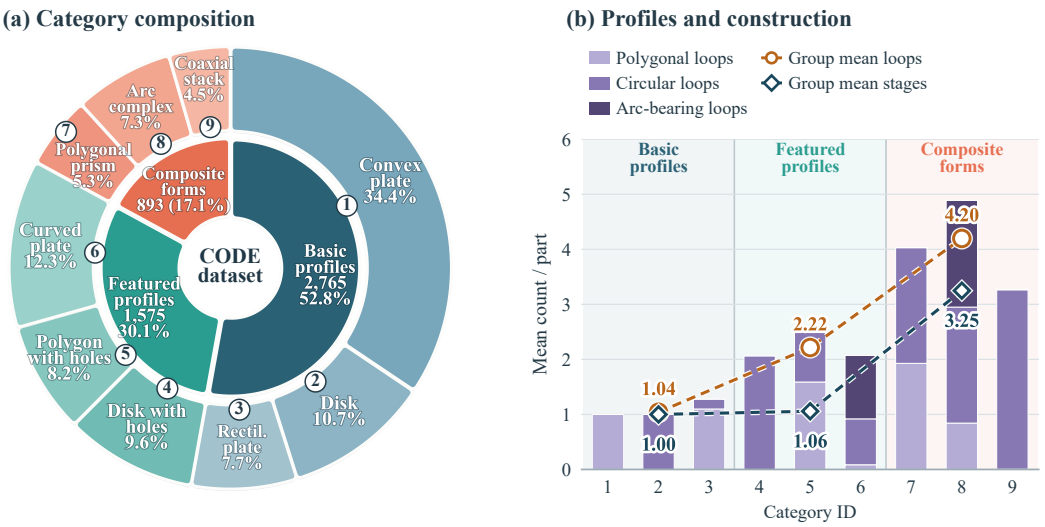


Figure 4: Category composition and structural characteristics of CODE. (a) Nine categories organized into three construction-focused groups; sector numbers identify the categories in (b). (b) Stacked bars show the mean number of closed-profile inputs per sample in each category, broken down by profile type. Dashed lines show sample-weighted group means of profile inputs and extrusion stages, with one point at each group center. Connecting segments serve only as visual guides.

5 Experiments

5.1 Experimental Setup

DraftCoder is built upon Qwen3-VL-8B-Instruct. We perform one epoch of SFT using LoRA ($r = 8$, $\alpha = 32$), with a learning rate of 10^{-4} and a maximum sequence length of 16,384 tokens. For GRPO, we use 96 distinct reward-eligible training samples for 48 optimization steps, with a learning rate of 10^{-6}, four candidate programs per input, a KL coefficient of 0.02, and a maximum generation length of 12,288 tokens.

Our baselines comprise three open-source vision-language models—Qwen3-VL-8B-Instruct, GLM-4.6V-Flash, and Kimi-VL-A3B-Instruct—and two closed-source models, GPT-5.6-SOL and Claude Opus-5.

5.2 Main Results

End-to-end generation reliability. As shown in Table 2, DraftCoder achieves a Scorable DXF Rate of 86.26%, exceeding GPT-5.6-SOL's 41.40% by 44.86 percentage points. The corresponding rates for Qwen3-VL-8B, GLM-4.6V-Flash, and Kimi-VL-A3B range from 0 to 0.67%. Observed failure cases include repetitive or truncated outputs, incorrect `ezdxf` API calls, and incomplete output procedures. Figure 5(b) illustrates an unsuccessful tube-generation case: GPT-5.6-SOL does not deliver a DXF file, whereas DraftCoder produces the corresponding drawing.

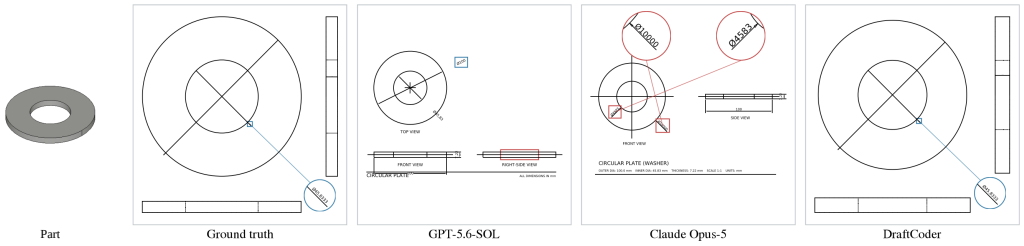
View completeness and geometric representation. Beyond generation reliability, DraftCoder achieves View Coverage and Geometry Fidelity scores of 79.22 and 21.20, compared with 21.74 and 3.62 for GPT-5.6-SOL. Its Complete View Rate reaches 70.32%, compared with 2.50% for GPT-5.6-SOL. The tube example in Figure 5(b) illustrates coverage relative to the benchmark's three-view reference: Claude Opus-5 omits one longitudinal view, while DraftCoder retains the end view and both longitudinal views. Figure 5(a) further reveals a difference in local geometric detail: GPT-5.6-SOL omits the hidden edges of the through-hole in the right-side view, whereas DraftCoder preserves the corresponding reference features.

Native dimension annotations. DraftCoder obtains an Annotation Quality score of 42.97 and a Complete Annotation Match Rate of 41.16%, compared with 2.42 and 0.33% for GPT-5.6-SOL.

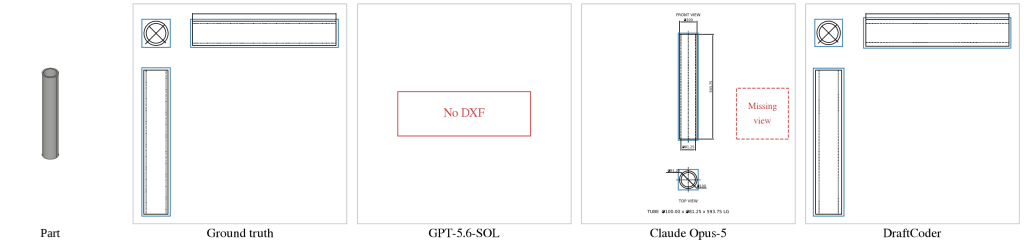
Table 2: Main results on the common evaluation set. All metrics are higher-is-better. Bold indicates the best available result in each column; a dash denotes a pending result.

Model	Scorable DXF Rate (%)	Structural Validity	View Coverage	Geometry Fidelity	Dimensional Fidelity	Annotation Quality	Layout Consistency	Complete Annotation Match Rate (%)	Complete View Rate (%)	Layout Acceptance Rate (%)
Open-source VLMs										
Qwen3-VL-8B	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GLM-4.6V-Flash	0.67	0.67	0.26	0.03	0.35	0.00	0.04	0.00	0.00	0.00
Kimi-VL-A3B	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Closed-source VLMs										
GPT-5.6-SOL	41.40	32.56	21.74	3.62	17.45	2.42	8.57	0.33	2.50	1.83
Claude Opus-5	—	—	—	—	—	—	—	—	—	—
DraftCoder (Ours)	**86.26**	**85.59**	**79.22**	**21.20**	**21.85**	**42.97**	**53.08**	**41.16**	**70.32**	**31.03**

The six quality scores, from Structural Validity to Layout Consistency, use a 0–100 scale. Evaluation uses 1,041 samples; Complete Annotation Match Rate uses the 911 applicable samples.



(a) Washer.



(b) Tube.

Figure 5: Qualitative comparisons on a washer and a tube. Columns show the input part, ground truth, and outputs from three models. Callouts highlight diameter display values, hidden edges, and view coverage relative to the benchmark's three-view reference. "No DXF" denotes failure to deliver a DXF file.

Similar geometric outlines do not necessarily imply correct dimension annotations. In the washer example in Figure 5(a), Claude Opus-5 produces the annular outline but displays incorrect diameter values, whereas DraftCoder preserves the key diameter value shown in the reference. Together, these results demonstrate DraftCoder's advantages in delivering scorable DXF files, reproducing the required reference views, and generating native dimension annotations.

5.3 Ablation Study

We compare the SFT model, the two reward-ablation variants, and the full DraftCoder model to examine the effects of execution feedback. Table 3 reports end-to-end delivery and completeness, while Table 4 provides a supplementary analysis of drawing quality on the fixed subset of 768 cases successfully generated by all four variants.

Effect of execution feedback. Compared with SFT, DraftCoder improves DXF delivery, view completeness, and dimension-annotation performance. Complete Annotation Match Rate increases from 37.32% to 41.16%, a relative improvement of 10.29%, while Layout Acceptance Rate improves by

Table 3: Reward ablations on end-to-end delivery and completeness.

Model	Scorable DXF Rate (%)	Complete View Rate (%)	Complete Annotation Match Rate (%)	Layout Acceptance Rate (%)
DraftCoder-SFT	83.96 (874/1041)	66.76 (695/1041)	37.32 (340/911)	28.43 (296/1041)
DraftCoder w/o RL_G	87.42 (910/1041)	**70.89 (738/1041)**	37.10 (338/911)	28.82 (300/1041)
DraftCoder w/o $\mathrm{RL}_{P,A}$	**87.80 (914/1041)**	70.61 (735/1041)	38.53 (351/911)	28.05 (292/1041)
DraftCoder	86.26 (898/1041)	70.32 (732/1041)	**41.16 (375/911)**	**31.03 (323/1041)**

Table 4: Reward ablations on drawing quality over the same jointly successful cases.

Model	Structural Validity	View Coverage	Geometry Fidelity	Dimensional Fidelity	Annotation Quality	Layout Consistency
DraftCoder-SFT	99.48	92.93	25.97	26.39	51.74	64.57
DraftCoder w/o RL_G	99.35	**94.18**	26.27	26.55	51.60	**65.00**
DraftCoder w/o $\mathrm{RL}_{P,A}$	99.61	93.84	**27.02**	27.03	51.50	64.31
DraftCoder	**99.74**	93.45	26.69	**27.14**	**53.95**	64.66

9.12% relative. These gains suggest that execution feedback helps the model produce more scorable DXF outputs while better preserving required annotations and satisfying layout requirements, extending improvements in program generation to the quality of the resulting drawings.

Dimension–annotation rewards. The benefits of this reward group are clearest in annotation quality and completeness. Compared with the variant without dimension–annotation rewards, DraftCoder improves Complete Annotation Match Rate by 6.84% relative and raises Annotation Quality from 51.50 to 53.95 on the common success subset. Although the ablated variant achieves a higher end-to-end Scorable DXF Rate and higher Geometry Fidelity on the common success subset, the full model better preserves dimension annotations. This contrast suggests that recovering geometric contours alone does not ensure that dimensional information and its associations with geometric entities are correctly retained, motivating explicit constraints on annotation quality.

Geometry feedback. On the common success subset, DraftCoder improves Geometry Fidelity from 26.27 to 26.69 and Dimensional Fidelity from 26.55 to 27.14 compared with the variant without the explicit geometry reward and its downstream gate. These gains are consistent with the role of this feedback configuration in assessing executed DXF contours and geometric relationships under a shared alignment, providing constraints on geometric structure and dimensional consistency.

6 Conclusion

In this work, we introduce DraftCoder, which combines supervised fine-tuning with reinforcement learning guided by a Hierarchical Drawing-Consistency Reward (HDCR) to generate executable programs from part images and descriptions of key dimensions. These programs jointly represent multi-view geometry and native dimension annotations in editable DXF drawings. We construct CODE, a dataset of 5,233 samples spanning nine categories of part structures, together with an evaluation benchmark, and will publicly release the dataset to support further research. Experiments show that DraftCoder improves scorable DXF delivery, view completeness, and dimension annotation quality, while ablation studies further support the effectiveness of execution feedback.

Future work will incorporate design and manufacturing requirements to extend generation to engineering annotations such as tolerances and surface roughness, enabling a more comprehensive representation of engineering information.

REFERENCES

Anna C. Doris, Md Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. CAD-Coder: An open-source vision-language model for computer-aided design code generation. *arXiv preprint arXiv:2505.14646*, 2025. doi: 10.48550/arXiv.2505.14646. URL https://arxiv.org/abs/2505.14646.

Anna C. Doris, Jacob Thomas Sony, Ghadi Nehme, Era Syla, Amin Heyrani Nobari, and Faez Ahmed. CADBench: A multimodal benchmark for AI-assisted CAD program generation. *arXiv preprint arXiv:2605.10873*, 2026. URL https://arxiv.org/abs/2605.10873.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-Coder: Text-to-CAD generation with chain-of-thought and geometric reward. In *Advances in Neural Information Processing Systems*, volume 38, pp. 59765–59789, 2025. doi: 10.52202/085713-1999.

Changqi He, Shuhan Zhang, Liguo Zhang, and Jiajun Miao. CAD-Coder: Text-guided CAD files code generation. *arXiv preprint arXiv:2505.08686*, 2025. doi: 10.48550/arXiv.2505.08686. URL https://arxiv.org/abs/2505.08686.

Aditya Joglekar, Amit Regmi, Kenji Shimada, and Levent Burak Kara. Ortho2CAD: 3D CAD generation from orthographic drawings using vision language models. *arXiv preprint arXiv:2607.08891*, 2026. doi: 10.48550/arXiv.2607.08891. URL https://arxiv.org/abs/2607.08891.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems*, volume 37, pp. 7552–7579. Curran Associates, Inc., 2024. doi: 10.52202/079017-0242.

Mingi Kim, Yongjun Kim, and Hyungki Kim. Drawing-Recode: Annotation grounding for parametric CAD code generation from raster 2D CAD drawings. *arXiv preprint arXiv:2607.27558*, 2026. doi: 10.48550/arXiv.2607.27558. URL https://arxiv.org/abs/2607.27558.

Dimitrios Mallis, Marco Wang, Ahmet Serdar Karadeniz, Elisa Ricci, Anis Kacem, and Djamila Aouada. Text-to-CAD evaluation with CADTests. *arXiv preprint arXiv:2605.07807*, 2026. URL https://arxiv.org/abs/2605.07807.

Ke Niu, Yuwen Chen, Haiyang Yu, Zhuofan Chen, Xianghui Que, Bin Li, and Xiangyang Xue. PHT-CAD: Efficient CAD parametric primitive analysis with progressive hierarchical tuning. *arXiv preprint arXiv:2503.18147*, 2025a. URL https://arxiv.org/abs/2503.18147.

Ke Niu, Zhuofan Chen, Haiyang Yu, Yuwen Chen, Teng Fu, Mengyang Zhao, Bin Li, and Xiangyang Xue. CReFT-CAD: Boosting orthographic projection reasoning for CAD via reinforcement fine-tuning. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-2765.

Ke Niu, Haiyang Yu, Zhuofan Chen, Zhengtao Yao, Weitao Jia, Xiaodong Ge, Jingqun Tang, Benlei Cui, Bin Li, and Xiangyang Xue. CME-CAD: Heterogeneous collaborative multi-expert reinforcement learning for CAD code generation. *arXiv preprint arXiv:2512.23333*, 2025c. doi: 10.48550/arXiv.2512.23333. URL https://arxiv.org/abs/2512.23333.

Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. SketchGen: Generating constrained CAD sketches. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Vladislav Pyatov, Gleb Bobrovskikh, Saveliy Galochkin, Nikita Boldyrev, Oleg Voynov, Alexander Filippov, Gonzalo Ferrer, Peter Wonka, and Evgeny Burnaev. CADFS: A big CAD program dataset and framework for computer-aided design with large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10176–10186, 2026.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse engineering CAD code from point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9801–9811, October 2025.

Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P. Adams. Vitruvion: A generative model of parametric CAD sketches. In *International Conference on Learning Representations*, 2022.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-CAD generation through infusing visual feedback in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 65326–65345. PMLR, 2025.

Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 Gallery: A dataset and environment for programmatic CAD construction from human design sequences. *ACM Transactions on Graphics*, 40 (4):54:1–54:24, 2021. doi: 10.1145/3450626.3459818.

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

Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. CAD-MLLM: Unifying multimodality-conditioned CAD generation with MLLM. *arXiv preprint arXiv:2411.04954*, 2024. URL https://arxiv.org/abs/2411.04954.

Haozhe Zhang, Kaichen Liu, Miaomiao Chen, Lei Li, Shaojie Yang, Cheng Peng, and Hanjie Chen. BenchCAD: A comprehensive, industry-standard benchmark for programmatic CAD. *arXiv preprint arXiv:2605.10865*, 2026. URL https://arxiv.org/abs/2605.10865.

A GRPO Optimization Details

For an image–text input $x = (I, T)$ drawn from the RL training set $\mathcal{D}_{\mathrm{RL}}$, the sampling policy π_{old} generates K candidate programs $\{C_k\}_{k=1}^{K}$. Each candidate receives the HDCR reward r_k defined in Equation 3. We compute $\bar{r}(x) = K^{-1} \sum_{j=1}^{K} r_j$ and $\hat{A}_k = (r_k - \bar{r}(x))/(\sigma_B + \delta)$, where σ_B is the reward standard deviation across the current sampled generation batch and δ ensures numerical stability.

Let $c_{k,t}$ denote token t of candidate k and n_k its number of valid generated tokens. The token-level probability ratio between the current and sampling policies is $\rho_{k,t} = \pi_\theta(c_{k,t} \mid x, c_{k,<t})/\pi_{\mathrm{old}}(c_{k,t} \mid x, c_{k,<t})$. We minimize

$$\mathcal{L}_{\mathrm{GRPO}} = -\mathbb{E}_{\substack{x \sim \mathcal{D}_{\mathrm{RL}} \\ C_1, \ldots, C_K \sim \pi_{\mathrm{old}}(\cdot|x)}} \left[\frac{1}{K} \sum_{k=1}^{K} \frac{1}{n_k} \sum_{t=1}^{n_k} \left(\min \left[\rho_{k,t} \hat{A}_k,\ \mathrm{clip}(\rho_{k,t}, 1-\epsilon, 1+\epsilon) \hat{A}_k \right] - \beta \widehat{D}_{k,t}^{\mathrm{KL}} \right) \right]. \tag{4}$$

Here, ϵ controls probability-ratio clipping, β is the KL regularization coefficient, and $\widehat{D}_{k,t}^{\mathrm{KL}}$ is the token-level KL estimate relative to the frozen SFT reference policy π_{ref}. The expectation is over training inputs and candidate programs sampled conditionally from π_{old}. The loss is averaged over valid tokens within each candidate and then across candidates. The sampling policy π_{old} supplies the probability-ratio denominator, while π_{ref} remains fixed throughout RL.

B Additional Dataset Details

B.1 Category-wise Engineering-Drawing Characteristics

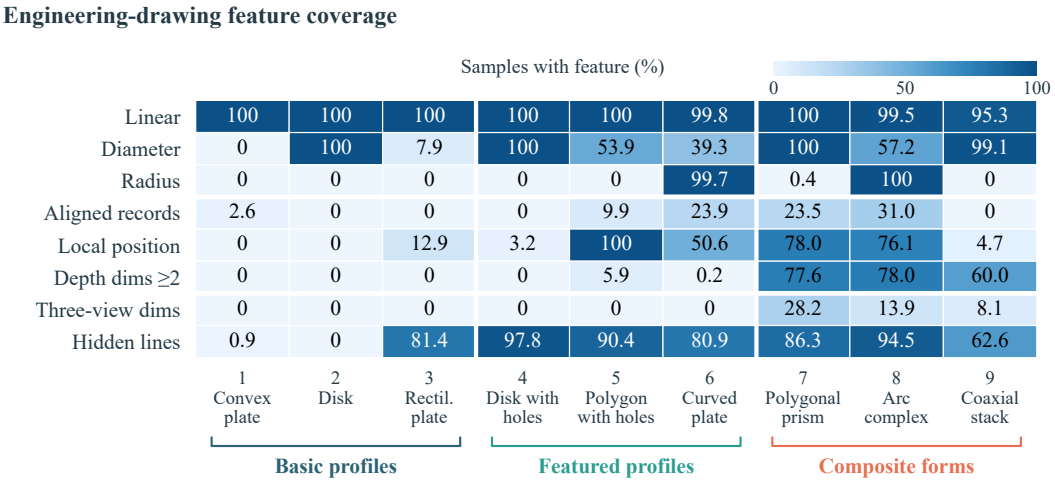


Figure 6: Engineering-drawing feature coverage by category in CODE. Rows represent dimension types, annotation characteristics, and hidden-line presence; columns represent the nine part categories. Cell values indicate the percentage of valid samples in each category exhibiting the corresponding feature, with darker blue indicating higher coverage.

The engineering drawings exhibit distinct dimensioning requirements across categories (Figure 6). Linear dimensions are prevalent in all nine categories. Disk drawings use diameter dimensions to specify circular size, whereas curved plates and arc-bearing composite parts commonly include radius dimensions, with coverage of 99.7% and 100%, respectively. Local-position dimensions occur in all valid polygon-with-holes samples, indicating that these drawings specify not only feature sizes but also their relative locations. Hidden lines appear in 97.8% of disks with holes and 90.4% of polygons with holes, reflecting the need to represent occluded structures.

Multiple depth/thickness annotations and a broader distribution of dimensions across views are particularly prominent in the Composite forms group. In Categories 7–9, 60.0%–78.0% of samples contain at least two depth/thickness dimension records, compared with no more than 5.9% in Categories 1–6. Drawings with dimensions in all three views occur only in Categories 7–9. These composite categories nevertheless have distinct emphases: local-position coverage exceeds 76% in both the polygonal and arc-bearing composite categories, whereas the circular composite category emphasizes diameter and depth information, with diameter coverage reaching 99.1%.

B.2 Feature Definitions and Computation

Coverage calculation. For category c and feature k, let $\mathcal{V}_{c,k}$ be the set of samples with valid records for that statistic, and let $z_{i,k} \in \{0, 1\}$ indicate whether sample i satisfies the corresponding condition in Table 5. The coverage reported in Figure 6 is

$$P_{c,k} = 100 \frac{\sum_{i \in \mathcal{V}_{c,k}} z_{i,k}}{|\mathcal{V}_{c,k}|}. \tag{5}$$

Coverage measures the fraction of samples exhibiting a feature, rather than the number of annotations per drawing.

Table 5: Sample-level conditions for the eight engineering-drawing features. Each condition defines when $z_{i,k} = 1$.

Feature	Sample-level condition
Linear	At least one native linear `DIMENSION` entity in the final DXF, with base type 0 or 1.
Diameter	At least one native diameter `DIMENSION` entity (base type 3).
Radius	At least one native radius `DIMENSION` entity (base type 4).
Aligned records	At least one emitted `dims.json` record with `orient=aligned`. This records the output annotation form rather than an angular dimension.
Local position	At least one emitted linear dimension linked to a planning role in {`hole_x`, `hole_y`, `rib_x`, `rib_y`}.
Depth dims ≥ 2	At least two emitted linear dimensions linked to the `depth` planning role. The threshold applies to the number of records, not to a measured depth value.
Three-view dims	At least one emitted dimension assigned to each of the front, top, and left views.
Hidden lines	At least one model-space geometric entity on the `HIDDEN` layer or with a resolved linetype beginning with `HIDDEN`, after the line-length filter described below.

Record extraction. DXF dimension base types are obtained by masking the `dimtype` value with 15. Hidden-line statistics consider model-space `LINE`, `ARC`, `CIRCLE`, `ELLIPSE`, `SPLINE`, `LWPOLYLINE`, and `POLYLINE` entities. `BYLAYER` linetypes are resolved through the layer definition, `LINE` entities of length at most 10^{-6} drawing units are excluded, and dimension-rendering blocks are not traversed. For record-based statistics, actual emitted dimension records are uniquely linked to `view_plan.json` by label and compatible type. Counts and dimension families are reconciled with the final DXF; view assignments and semantic roles are taken from the linked records. In particular, aligned records are identified by the emitted orientation field, not solely by DXF base type 1.

Valid samples. Linear, diameter, radius, and hidden-line statistics use all 5,233 readable final DXFs. Aligned-record, local-position, depth, and annotated-view statistics use the 5,232 samples with valid linked records. One polygon-with-holes sample lacks these records, so its category denominator is 426 for record-based statistics and 427 for DXF-based statistics. Observed zero-annotation samples remain in the applicable denominators.

Relation to the overview statistics. Table 1 reports annotation totals as well as sample percentages. For an annotation type or role r, its total is $T_r = \sum_{i \in \mathcal{V}_r} n_{i,r}$, where $n_{i,r}$ is the number of corresponding entities or emitted records in sample i. Thus, depth and local-position totals sum role-specific linear records, whereas Figure 6 thresholds their per-sample counts. These roles are subsets of linear dimensions, not additional dimension types. The overall hidden-line percentage uses Equation 5 over all categories. For the percentage of drawings dimensioned in at least two views, the same equation uses $z_i = \mathbf{1}[a_i \geq 2]$, where

$$a_i = \sum_{v \in \{\text{front,top,left}\}} \mathbf{1}[n_{i,v} > 0] \tag{6}$$

counts the views containing emitted dimensions, with $n_{i,v}$ denoting the number assigned to view v.

B.3 Construction Feature Extraction

Operation recording. We execute the source CadQuery program against a lightweight replacement of the CadQuery interface that records construction operations without generating boundary geometry. The recorder intercepts sketch-plane definitions, profile creation, extrusion, translation, and Boolean combination. Each extrusion creates a construction record containing the sketch frame, local profile coordinates, signed extrusion distance, and symmetric-extrusion flag. Subsequent Boolean calls identify the corresponding operand's union, cut, or intersection role. Accumulated translations are applied to the sketch origin so that profile coordinates remain registered to the model coordinate system.

Profile geometry and measurement candidates. Ordered profile segments are classified as horizontal, vertical, or oblique. The extractor consolidates repeated horizontal and vertical extents into candidate dimensions and retains the lengths and endpoints of oblique segments. Extended parsers preserve multiple closed loops and circular profiles. Within the supported profile patterns, the largest-area polygonal loop is treated as the outer boundary; additional loops are retained as internal profiles. Circular features store their centers, radii, diameters, and positional offsets relative to a profile datum or an outer-circle center. Three-point arc calls preserve their defining points and recover the circle center, radius, and traversal direction. The extended parsers also remove consecutive duplicate points and redundant collinear vertices, improving correspondence with the exported geometry.

Structured output and scale. The resulting `features.json` contains source identification, a common scale factor, and ordered construction records with geometric references and dimension candidates. The largest recorded profile-edge length or outer-circle diameter is scaled to 100 mm, with the same factor applied to the STEP geometry. Signed extrusion distance and symmetry are stored separately; full thickness is resolved during view planning. A construction record denotes an extrusion input, not necessarily a connected solid in the final B-rep.

B.4 View Planning and 3D Measurement References

Front-view selection and view assignment. The planner associates each sketch normal with its dominant coordinate axis. For each axis, it accumulates the numbers of extracted widths, heights, oblique lengths, and outer-profile diameters. The axis with the largest count becomes the front-view direction. Ties are resolved by accumulated profile area, using polygon bounding-box areas or circular areas. The two remaining axes define the other orthographic views. Profile and circular-feature dimensions are assigned to the view facing their sketch plane. Extrusion depth is assigned to a perpendicular viewing direction, with a fixed preference order of top, left, then front, so that the extrusion extent is not collapsed by projection.

Pattern-specific dimension rules. Full circular profiles and circular holes receive diameter dimensions; extracted arcs receive radius dimensions. Nonzero hole-position offsets are measured from the lower-left profile datum or, for circular layouts, the outer-circle center. Rectilinear profiles use edge-wise dimensions, retaining distinct steps even when their lengths are equal. Selected Boolean polygon features use projected width and height measurements, with coincident projected extents consolidated. Repeated arc radii are represented once per construction record and loop role. For supported coaxial cuts, the planner distinguishes through openings from blind features and includes the latter's depth.

Constructing model-space anchors. Let $\mathbf{o}$, $\hat{\mathbf{x}}$, and $\hat{\mathbf{n}}$ denote the sketch origin, unit in-plane horizontal direction, and unit normal. A local profile point (u, v) is mapped to

$$\mathbf{p} = \mathbf{o} + u\hat{\mathbf{x}} + v(\hat{\mathbf{n}} \times \hat{\mathbf{x}}) + \delta\hat{\mathbf{n}}, \tag{7}$$

where δ selects the supporting plane. Profile anchors use the sketch plane for one-sided extrusions and an end face for symmetric extrusions. A depth measurement connects the appropriate end faces, giving $|t|$ for a one-sided extrusion and $2|t|$ for a symmetric extrusion with signed distance t. Circle centers and representative arc points are transformed using the same frame. The resulting `view_plan.json` retains the source construction record, dimension type and value, target view, and endpoint or curve-reference anchors.

B.5 Projection and Editable Annotation Construction

Orthographic projection and coordinate registration. The STEP model is imported into FreeCAD, where a TechDraw projection group creates front, top, and left views using first-angle projection. The group is scaled and positioned on an A4 landscape page, with hidden-edge computation enabled. For model-space anchor $\mathbf{p}$, its page position in view v is

$$\mathbf{q}_v = s\big[\pi_v(\mathbf{p}) - \mathbf{c}_v\big] + \mathbf{t}_v, \tag{8}$$

where π_v is the view projection, $\mathbf{c}_v$ is the projected bounding-box center, s is the drawing scale, and $\mathbf{t}_v$ is the page position of the view center. Bounding-box corners supplement model vertices when estimating $\mathbf{c}_v$, avoiding center errors for circular parts with sparse topological vertices.

Geometric references and annotation placement. Linear dimensions reference matching model vertices when available. Hole centers, datum points, and Boolean-modified extents without suitable model vertices use cosmetic vertices in the projected view, supporting editable native TechDraw distance annotations. Diameter references are selected by radius agreement and center proximity; arc-radius references additionally use a representative midpoint to distinguish curves with similar radii. Annotation placement uses exterior offsets, staggered depth-dimension positions, and distributed circular leader directions. Measured and planned values are compared and discrepancies are logged; collapsed cosmetic-vertex measurements are removed.

Intermediate drawing records. The stage saves the FCStd document together with `dims.json` and `hidden_edges.json`. Dimension records carry the view, measurement type, physical value, and page-coordinate endpoints or curve parameters, including annotation placement. Hidden-edge records preserve the projected geometry, with segments overlapping visible geometry filtered out.

B.6 Drawing-Program Generation and Sample Assembly

Geometry conversion. The final TODXF02 route extracts visible edges directly from the FCStd TechDraw views and combines them with the hidden-edge and dimension records. A visible-edge point (x, y) is mapped to page coordinates as $(x + g_x + v_x, -y + g_y + v_y)$, where (g_x, g_y) is the projection-group anchor and (v_x, v_y) is the view offset. The extracted edge coordinates already include the page scale. Analytic lines, circles, and arcs are retained as corresponding primitives. Projected spline representations are simplified to segments when nearly straight, recovered as circular primitives when fitting succeeds, and otherwise discretized. The assembler removes degenerate segments, deduplicates coincident visible segments, and places visible and hidden geometry on the `OUTLINE` and dashed `HIDDEN` layers, respectively.

Native dimensions and executable targets. Horizontal, vertical, aligned, diameter, and radius records are converted using native ezdxf dimension APIs. Endpoints, centers, leader angles, and dimension-line placements are taken from the exported records. For aligned dimensions, the signed perpendicular offset is recovered from the endpoints and stored base point. Displayed dimension text retains the recorded physical value independently of the drawing scale. A companion generator embeds the geometry and annotation parameters into a self-contained Python program that reconstructs the DXF using ezdxf, without FreeCAD or external intermediate JSON files. Previews are rendered directly from the resulting DXF, including its native dimension entities.

Export checks and multimodal packaging. Before export, radial records are checked against projected curves with center and radius tolerances of 0.5 and 0.15 page millimeters. A linear record is flagged when both endpoints are over 5 page millimeters from candidate geometry. Normal export blocks radial or excessive linear mismatches and reports isolated discrepancies. Description templates summarize profile shapes, available dimensions, holes or cutouts, thickness, and oblique edges in construction-record order. The packager adds a drawing instruction and pairs each description with its isometric image and Python target. Stable IDs link samples to separate category and split metadata. Entries missing an image, description, or program are omitted. The validator checks image availability and Python syntax and supports sampled execution to verify DXF generation.