INTENTCAD: LEARNING TEXT-TO-CAD FROM THE INTENT BEHIND THE SHAPE

Anonymous authors
Paper under double-blind review

ABSTRACT

A CAD program that reproduces the right shape can still hide how the part was designed. The same mounting plate can be written as four separate circle cuts at literal coordinates or as one hole pattern with a shared diameter, and many text-to-CAD training pairs take the first form, with descriptions that leave the part's purpose unstated. We ask whether making this design information explicit in training improves generation even when test prompts describe geometry alone. To test this, we construct IntentCAD, 446K text–program pairs from 223K parts, whose programs organize operations into features with named parameters and explicit relations, are verified against the source geometry, and are paired with descriptions that also state the part's use and design intent. In controlled experiments with a 4B model on Text2CAD-Bench, normalizing programs and descriptions lowers Chamfer distance by 27% on average. Adding design intent, which no test prompt mentions, lowers it by a further 11% and raises IoU on all four splits. The resulting model also produces valid programs more often than four general-purpose LLMs, with an average invalid rate of 11.0% against 20.7–63.6%. These results suggest that what a text-to-CAD model learns depends on the design information its training data exposes, beyond the geometry it reproduces.

1 INTRODUCTION

Asked for a plate that mounts a small motor and lets it slide to tension a belt, an engineer does not simply draw a plate with holes. The bolt holes become one symmetric pattern with a shared diameter, and the slots take their length from the required travel. A parametric CAD model records such decisions as features, parameters, and relations, which is what lets the part be understood, reused, and modified. Text-to-CAD generation aims to produce such models from natural language, and LLMs that write CAD programs (Khan et al., 2024; Li et al., 2025; Xie and Ju, 2025), further trained with geometric rewards (Guan et al., 2025; Kolodiazhnyi et al., 2026; Li et al., 2026b), now produce accurate shapes for simple parts.

The same plate can be written in very different ways. A program can cut four circles at literal coordinates, or it can declare one hole pattern whose diameter and spacing are named parameters. Both reproduce the geometry, and only the second exposes the relations behind it. Descriptions differ in the same way: one lists the shapes, and another also says what the part is for. Many text-to-CAD training pairs take the first form on both sides, converting construction sequences into programs and describing the resulting shape (Khan et al., 2024; Xie and Ju, 2025). A model trained on such pairs sees the geometry of a design and little of its structure or purpose.

This leads to the question we study: *what design information, beyond geometry, should the training data of a text-to-CAD model contain?* We make the question testable by asking whether information added only to the training data still helps when test prompts describe geometry alone. We hypothesize that it does. The purpose of a part can act like privileged information (Vapnik and Vashist, 2009): it is present when the model is trained, absent when the model is used, and still able to shape what the model learns. We study this effect on learning to generate, measured by geometric accuracy and validity.

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.

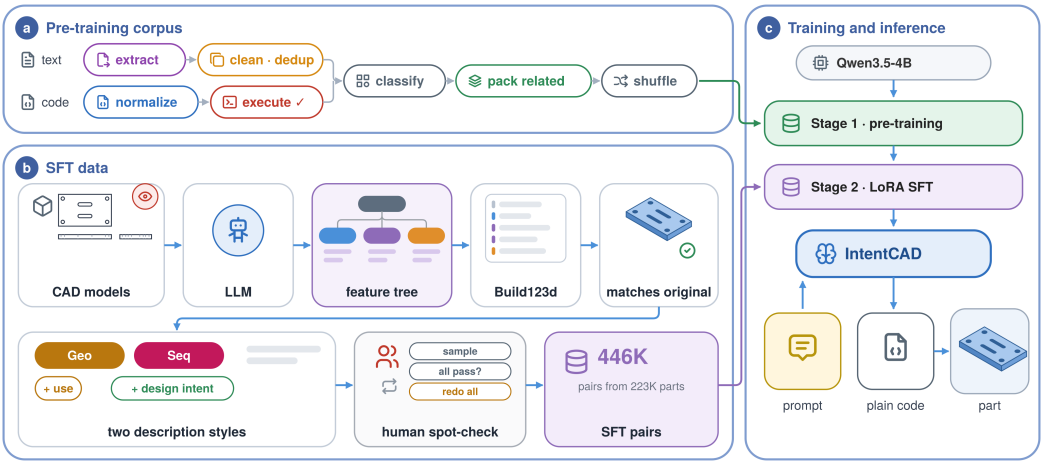

Figure 1: Overview. (a) Text sources are extracted, cleaned, and deduplicated, and code is normalized and executed; documents are then classified, and related documents are packed into the same training sequence. (b) Each collected model is converted to STEP with three standard views and reviewed by humans. An LLM reconstructs it as a Build123d program that begins with a global feature tree, kept only if it matches the original model. Each program is paired with a geometric and a sequence description that also state the part's use and design intent, checked by repeated human sampling. The tree is removed from the program before training. (c) The model is trained in two stages and outputs plain Build123d code.

To test the hypothesis, we build IntentCAD, a dataset of 446K text–program pairs from 223K parts in existing CAD datasets and industrial sources. Each part is reconstructed into a program that organizes operations into features with named parameters and explicit relations, and the program is kept only if it reproduces the source geometry. Each program is paired with two descriptions in the prompt styles of Text2CAD-Bench, both of which also state the part's use and design intent. We pre-train a 4B model on CAD documentation and design materials and fine-tune it on these pairs, adding one kind of information at a time. Figure 1 gives an overview.

The results support the hypothesis. Normalizing the programs and the descriptions lowers Chamfer distance by 27% on average across the four splits of Text2CAD-Bench, and adding design intent, which no test prompt mentions, lowers it by a further 11% and raises IoU on every split. The resulting model, also named IntentCAD, produces valid programs more often than four general-purpose LLMs on every split and is far more geometrically accurate than three specialized text-to-CAD models.

Our contributions are as follows:

1. We identify design information that text-to-CAD training data leaves implicit and build IntentCAD, 446K pairs of normalized parametric programs and descriptions that state use and design intent, reconstructed from 223K parts and verified against the source geometry (Section 3).
2. Through controlled comparisons, we show that program normalization, description normalization, and design intent each contribute, and that design intent helps on test prompts that do not contain it (Section 4.2).
3. The resulting 4B model produces valid programs more reliably than four general-purpose LLMs on every split and is far more geometrically accurate than specialized text-to-CAD models (Section 4.3).

2 RELATED WORK

Text-to-CAD generation. Text2CAD (Khan et al., 2024) introduced text-to-CAD generation with multi-level prompts annotated on DeepCAD models and a transformer that outputs sketch-and-extrude

command sequences. Subsequent work represents the CAD model as code and builds on LLMs. CAD-Llama (Li et al., 2025) converts command sequences into structured parametric code for adaptive pre-training and instruction tuning; Text-to-CadQuery (Xie and Ju, 2025) fine-tunes open LLMs on CadQuery programs generated from Text2CAD annotations. A further line adds reinforcement learning: CAD-Coder (Guan et al., 2025) combines chain-of-thought with a Chamfer-distance reward, CADFusion (Wang et al., 2025) uses visual feedback on rendered outputs, cadrille (Kolodiazhnyi et al., 2026) applies online RL across point-cloud, image, and text inputs, ReCAD (Li et al., 2026a) and CAD-RL (Niu et al., 2025) extend RL to vision-language models, and FutureCAD (Li et al., 2026b) adds a B-rep grounding module for operations such as fillet and chamfer that act on selected faces and edges. These methods improve executability and geometric fidelity mainly through the model and the training objective. Our work uses a standard LLM with supervised training and focuses on the content of the training data.

CAD data and representations. Large CAD datasets supply the geometry and construction histories these models learn from. ABC (Koch et al., 2019) provides one million B-rep models; DeepCAD (Wu et al., 2021) and Fusion 360 Gallery (Willis et al., 2021) record sketch-and-extrude sequences; SketchGraphs (Seff et al., 2020) captures constraint graphs at the sketch level. Recent work converts or synthesizes executable programs at scale: CAD-Recode (Rukhovich et al., 2025) procedurally generates CadQuery programs, CADEvolve (Elistratov et al., 2026) evolves programs into about 1.3M scripts, Zero-to-CAD (Ataei et al., 2026) synthesizes about one million executable construction sequences with an LLM in a CAD feedback loop, CADmium (Govindarajan et al., 2026) re-annotates DeepCAD models with GPT-4.1 descriptions, and HistCAD (Dong et al., 2026b) represents parametric sequences with explicit constraints and annotates each model with its modeling process, geometric structure, and functional type. CADFS (Pyatov et al., 2026) is closest to our work. It cleans FeatureScript programs by making parameters explicit, unifying units, and replacing unstable entity identifiers with queries, and pairs each program with a construction-level description produced by an annotator–reviewer pipeline. Design intent has also been modeled at the sketch level, where constraint generation is aligned with the intended behavior of a sketch (Casey et al., 2025). Building on these efforts, we reconstruct parts from existing datasets and industrial sources into programs organized by features and their relations, verify each reconstruction against the source geometry, and extend descriptions with the part's use and design intent.

Evaluating text-to-CAD models. Text-to-CAD models are commonly evaluated by invalidity ratio, Chamfer distance, and IoU against a ground-truth shape. Text2CAD-Bench (Wang et al., 2026) stratifies parts by geometric complexity and pairs each part with a geometric and a sequence prompt. Other benchmarks examine design-level properties, such as whether constraints hold after dimension edits (Dong et al., 2026b), whether outputs meet functional requirements (Dong et al., 2026a; Singh et al., 2026), and whether they pass executable geometric tests (Mallis et al., 2026). We evaluate on the L1 and L2 tiers of Text2CAD-Bench (Section 4).

3 METHOD

Figure 1 summarizes our pipeline. We build two kinds of data for two training stages: a pre-training corpus that provides CAD and design knowledge (Section 3.2), and SFT pairs in which each part is reconstructed into a normalized parametric program and paired with descriptions in two styles (Section 3.3). We first specify the design information the data should carry (Section 3.1), and then describe training (Section 3.4).

3.1 DESIGN INFORMATION IN A CAD PROGRAM

We describe the structure of a single-part design by a feature tree $\mathcal{T}$. The root represents the part, and each child node is a feature f, such as a base body, a hole pattern, or a slot. A node stores

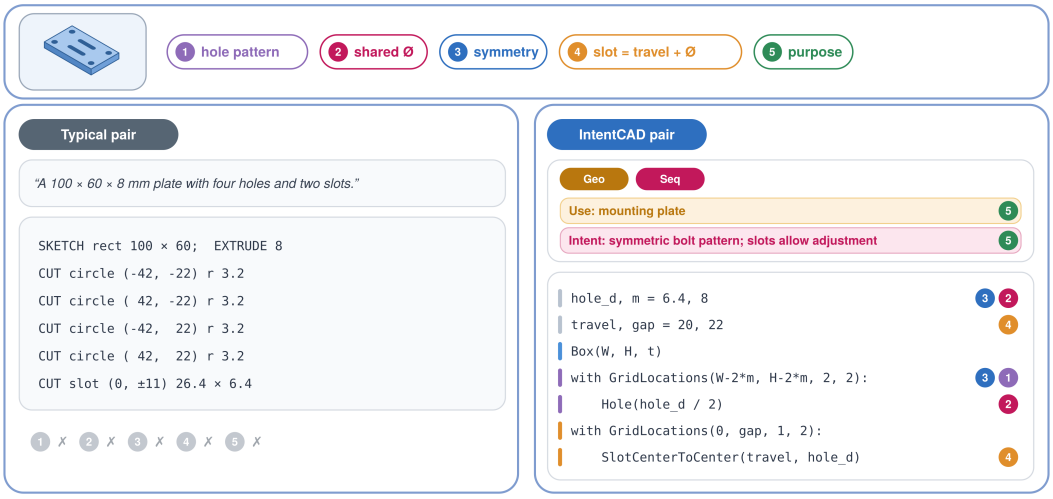

Figure 2: The same part as a typical training pair (left) and as an IntentCAD pair (right). Numbered badges mark five design decisions: the hole pattern, the shared diameter, the symmetric layout, the slot length that follows from the travel, and the purpose of the part. The typical pair records none of them; the IntentCAD pair records each in its program or descriptions. The example is illustrative.

named parameters Θ_f (e.g., hole diameter and edge margin), relations C_f to other parameters or features (e.g., symmetry or a derived length), and references A_f that attach the feature to faces or edges of earlier features; the order of the nodes gives the construction history. Beyond its structure, a design serves a purpose: the use the part supports and the intent behind its features, such as mounting, alignment, or clearance.

Existing text-to-CAD pairs carry part of this information. A construction sequence converted from DeepCAD, for instance, represents four mounting holes as four separate sketch-and-cut operations with literal coordinates. The geometry is complete, and the grouping into one feature, the shared diameter, the symmetric layout, and the attachment face all remain implicit; the paired description restates the resulting shape and leaves the purpose of the part unstated. Our data places the structure of $\mathcal{T}$ in the program and the purpose of the part in the description. Figure 2 shows both kinds of pairs for the same part. Table 1 compares the resulting dataset, IntentCAD, with existing CAD datasets along these dimensions.

3.2 PRE-TRAINING CORPUS

The pre-training corpus exposes the base model to parametric CAD and design knowledge before it sees paired data. It draws on documentation and API references of parametric CAD libraries, open-source modeling code, 3D and CAD design materials such as handbooks and standards, internal design documents, and general-purpose code. Text sources are extracted into a normalized format, cleaned, and deduplicated: each document is split at structural boundaries (top-level Python statements, headings, and code fences) into chunks of at most 8,192 characters, chunks shorter than 200 characters are dropped, and exact duplicates are removed after whitespace normalization. Code is normalized and enters the corpus only after it executes successfully. We then classify documents by topic, pack related documents into the same training sequence (Shi et al., 2024), and shuffle the sequences for training. Appendix A gives further processing details.

3.3 SFT DATA CONSTRUCTION

Collection. We collect CAD models and programs from DeepCAD (Wu et al., 2021) with augmentation, Zero-to-CAD (Ataei et al., 2026), CADmium (Govindarajan et al., 2026), CADFS (Pyatov et al., 2026), Seek-CAD (Li et al., 2026c), and ABC (Koch et al., 2019), together with real industrial

Table 1: Comparison of IntentCAD with existing CAD datasets that provide construction programs or paired text. ✓: provided; ❍: partially provided; ✗: not provided. Verification: programs are checked against a reference geometry (✓) or only for validity (❍). †Derived from DeepCAD.

Dataset	Size	Source	Program	Text	Features	Parameters	Relations	Intent	Verification	Human review
DeepCAD (Wu et al., 2021)	178K	real	command sequence	✗	✗	✗	✗	✗	✗	✗
Fusion 360 Gallery (Willis et al., 2021)	8.6K	real	sketch–extrude sequence	✗	✗	❍	✓	✗	✓	✗
Text2CAD (Khan et al., 2024)	170K	real†	command sequence	four detail levels	✗	✗	✗	✗	✗	✗
Text-to-CadQuery (Xie and Ju, 2025)	170K	real†	CadQuery (LLM-written)	from Text2CAD	✗	✗	✗	✗	❍	✓
CAD-Recode (Rukhovich et al., 2025)	1M	synthetic	CadQuery	✗	✗	✗	✗	✗	❍	✗
CADEvolve (Elistratov et al., 2026)	1.3M	synthetic	CadQuery	✗	❍	✗	❍	✗	❍	✗
HistCAD (Dong et al., 2026b)	170K	real	constrained sequence	process, structure, function	❍	✗	✓	❍	❍	✗
CADFS (Pyatov et al., 2026)	450K	real	FeatureScript	construction steps	✓	✗	❍	✗	✓	✗
IntentCAD (ours)	223K	real + synthetic	Build123d from feature tree	Geo + Seq styles	✓	✓	✓	✓	✓	✓

parts from private sources. All models are converted to STEP, and for each model we render three standard orthographic views. Human reviewers inspect the models visually at this stage and discard those with unreasonable topology or other defects.

Parametric reconstruction. For each model, GPT-5.2 reconstructs the part from the STEP geometry data and the three views as a program in the Build123d library (build123d Contributors, 2026); the STEP measurements are authoritative, and the views resolve direction, occlusion, and symmetry. Before writing any operation, it lays out the feature tree $\mathcal{T}$ of the whole part: the root is the final solid, and the ordered nodes cover the base body, additive and subtractive features, repeated or mirrored groups, and finishing features, each with its parent, role, reference plane or axis, dimensions, placement, and effect on the solid. The program then implements the tree. Dimensions are declared as named parameters at the top, and derived quantities such as radii and spacings are computed from them; operations follow a fixed order from the main body to additive, subtractive, and finishing features; each code stage carries the ID of the node it implements; and every operation must change the solid. The tree is written as comments at the beginning of the program and is removed before training, so each training target is plain executable Python. Many existing programs trace each outline with line and arc segments, for example `BuildLine` with `ThreePointArc`, and extrude it; in such programs a hole, a slot, or a pattern survives only as a fitted contour. We use Build123d because its builder contexts group the operations of each feature into one code block, which suits this organization. The program is executed and the resulting solid is compared with the original model; a reconstruction is kept only if $1 - \mathrm{IoU}$ with the original is at most 10^{-4}. Appendix C gives the prompts.

Descriptions. Following the two prompt styles of Text2CAD-Bench (Wang et al., 2026), GPT-5.2 writes a geometric description (Geo) and a sequence description (Seq) for each part from the STEP data, the three views, and the final program. Each description has three paragraphs. The first names a neutral part category, and a use only when the geometry supports it. The second states the design intent that the feature relationships establish, such as mounting, alignment, clearance, or symmetry, and leaves out any industry, material, load, or tolerance that the evidence does not show. The third gives the complete final geometry (Geo) or the effective modeling sequence with references to the feature IDs of the tree (Seq). Descriptions converted from command sequences list sketch entities one step at a time, so they are long and leave global features, such as a hole pattern or a symmetry, unstated; both of our styles state these features directly. Human reviewers then check a random sample; when the sample contains errors, all descriptions are re-screened and regenerated and a new sample is drawn, and the process repeats until every sampled description passes.

Training pairs. Each description is paired with the normalized program of its part. The resulting IntentCAD dataset contains 222,799 parts and 445,598 text–program pairs, two per part.

3.4 TRAINING

We initialize the IntentCAD model from Qwen3.5-4B (Qwen Team, 2026a). In the first stage, we continue pre-training all parameters on the corpus of Section 3.2 with the next-token objective for ten epochs at a learning rate of 5×10^{-6}. In the second stage, we fine-tune the model with LoRA (Hu et al., 2022) (rank 32, $\alpha = 64$, applied to all linear layers) on the SFT pairs, with a description of either style as input and the program as target; the loss is computed on the program only. We train for three epochs with AdamW, a learning rate of 1×10^{-4} with warmup and cosine decay, and a global batch size of 256 on 32 GPUs. Appendix B lists all settings. At inference, the model receives a text prompt and generates a plain Build123d program, which is executed to obtain the part.

4 EXPERIMENTS

4.1 SETUP

Benchmark. We evaluate on the L1 (single-feature parts) and L2 (multi-feature parts) tiers of Text2CAD-Bench (Wang et al., 2026), which contain 200 parts each. Each part has two prompts: a geometric prompt (Geo) that describes shapes and dimensions as a non-expert would, and a sequence prompt (Seq) that specifies modeling operations in the order a CAD professional would apply them. The prompts state no use or design intent, so these parts of our descriptions are used only in training.

Metrics. A generated program is valid if it executes without error and produces a non-empty, manifold solid. We report the invalid rate (IR, %), the Chamfer distance (CD, $\times 10^3$) between points sampled on the generated and reference surfaces, and the IoU between voxelizations at 128^3 resolution. Both shapes are normalized in scale and pose before CD and IoU are computed, and both metrics are averaged over valid outputs.

Data variants and reference models. All fine-tuned variants in Table 2 start from the same pre-trained checkpoint and use the fine-tuning configuration of Section 3.4. For reference, we compare with four general-purpose LLMs—DeepSeek-V4-Pro (DeepSeek-AI, 2026), Qwen3.6-Max-Preview (Qwen Team, 2026b), Kimi-k2.6 (Moonshot AI, 2026), and GLM-5.2 (Z.ai, 2026)—and with the three specialized text-to-CAD models evaluated in Text2CAD-Bench: Text2CAD (Khan et al., 2024), Text-to-CadQuery (Xie and Ju, 2025), and CADFusion (Wang et al., 2025). The LLMs and IntentCAD generate Build123d programs; the LLMs receive the same prompt, are sampled at temperature 0.6, and may retry up to three times when a program fails to execute. The specialized models generate in their native representations and run with their released code and default settings.

4.2 WHAT THE DATA CONTRIBUTES

Table 2 builds the model up step by step. The base model fails to produce a valid program for 78.0–95.5% of the prompts, and continual pre-training alone lowers this rate by 17.5 to 27.5 points. The remaining rows fine-tune the pre-trained model on the same parts with the same configuration. The first pairs programs that trace outlines with line and arc segments with descriptions converted from command sequences (Section 3.3); fine-tuning on these pairs brings the invalid rate down to 5.8–25.9%. The next steps replace them with normalized programs, with descriptions that state global features, and finally with descriptions that also state the part's use and design intent. Normalizing the programs lowers CD on three of the four splits, most on L1 Geo (12.56 to 8.25). Normalizing the descriptions then lowers CD and raises IoU on all four splits. Together, the two steps reduce CD by 50%, 27%, 10%, and 22% on L1 Geo, L1 Seq, L2 Geo, and L2 Seq; the invalid rate falls on L2 (19.40 to 17.00 and 25.87 to 18.50) and stays within 0.2 points on L1. Adding design intent lowers CD and raises IoU on all four splits, most on L1 Seq (CD 9.07 to 6.41, IoU 0.790 to 0.855); the invalid rate falls on L2 (17.00 to 13.00 and 18.50 to 17.00), stays at 6.00 on L1 Geo, and rises by

Table 2: Data variants on Text2CAD-Bench. The rows build the model up step by step, from the base model through continual pre-training to fine-tuning with increasing design information. IR in %, CD $\times 10^3$. ‡On L2, CD and IoU of the base model are averaged over only 12 (Geo) and 9 (Seq) valid outputs.

	L1 Geo			L1 Seq			L2 Geo			L2 Seq		
Variant	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑
Base (Qwen3.5-4B)‡	78.00	16.09	0.583	84.00	21.38	0.559	94.00	14.76	0.651	95.50	3.85	0.888
+ continual pre-training	50.50	11.74	0.653	66.50	17.10	0.728	72.50	13.90	0.624	77.50	22.36	0.683
+ fine-tuning on unnormalized pairs	5.80	12.56	0.712	6.28	12.51	0.676	19.40	16.57	0.614	25.87	17.24	0.634
+ program normalization	7.00	8.25	0.742	8.00	10.70	0.751	17.00	17.75	0.620	15.00	16.11	0.621
+ description normalization	6.00	6.25	0.789	6.50	9.07	0.790	17.00	14.89	0.663	18.50	13.51	0.678
+ design intent (IntentCAD)	6.00	6.19	0.833	8.00	6.41	0.855	13.00	13.78	0.698	17.00	12.42	0.707

Table 3: Comparison on Text2CAD-Bench. IR in %, CD $\times 10^3$. **Bold**: best among the models evaluated under our protocol. †Reported by Wang et al. (2026), which executes CadQuery, voxelizes at 256^3, and normalizes shapes to a unit bounding box.

	L1 Geo			L1 Seq			L2 Geo			L2 Seq		
Model	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑	IR↓	CD↓	IoU↑
DeepSeek-V4-Pro	13.5	6.96	0.827	21.7	**5.07**	**0.873**	29.8	**8.99**	**0.782**	43.3	13.11	0.744
Qwen3.6-Max-Preview	14.5	8.81	0.797	21.7	5.78	0.845	28.9	13.42	0.727	42.3	11.45	0.710
Kimi-k2.6	6.8	9.30	0.751	18.4	8.37	0.803	20.4	14.94	0.677	37.3	15.74	0.686
GLM-5.2	49.3	9.34	0.754	58.9	10.35	0.785	65.2	16.63	0.667	81.1	**8.62**	**0.757**
Text2CAD†	11.0	219.57	0.08	8.0	249.68	0.05	6.0	260.92	0.04	5.0	262.73	0.04
Text-to-CadQuery†	44.0	227.46	0.07	29.0	229.42	0.07	67.0	269.11	0.04	49.0	277.28	0.03
CADFusion†	60.0	229.62	0.03	65.0	212.17	0.05	70.0	236.63	0.03	64.0	242.68	0.03
IntentCAD (4B)	**6.0**	**6.19**	**0.833**	**8.0**	6.41	0.855	**13.0**	13.78	0.698	**17.0**	12.42	0.707

1.5 points on L1 Seq. Since the evaluation prompts contain no intent, this gain comes from what the model learns during training, in line with the role of privileged information.

4.3 COMPARISON WITH OTHER MODELS

Table 3 places IntentCAD among general-purpose LLMs and specialized text-to-CAD models. Among the models evaluated under our protocol, IntentCAD has the lowest invalid rate on all four splits (6.0–17.0%, against 6.8–81.1% for the LLMs) and the best CD and IoU on L1 Geo. On the other splits, its CD and IoU rank second to fourth of the five models. The specialized models, trained on DeepCAD-style data, produce geometry far from the reference: their CD exceeds 200 and their IoU stays below 0.1 on both tiers. Text2CAD executes reliably, with an invalid rate of 5–11%, and its outputs rarely match the requested shape.

5 CONCLUSION

We asked what design information, beyond geometry, the training data of a text-to-CAD model should contain. IntentCAD answers with programs that record how a part is organized and descriptions that state what it is for, reconstructed from existing datasets and industrial parts and verified against their geometry. Controlled comparisons show that each part of this information helps a 4B model, and that design intent helps on test prompts that never state it. The present study covers single parts built from regular features; free-form surfaces and multi-part assemblies are directions for future work.

REPRODUCIBILITY STATEMENT

Section 3 describes how IntentCAD is built: Section 3.2 and Appendix A cover the pre-training corpus, Section 3.3 covers the reconstruction, verification, description, and review steps, and Appendix C gives the full prompts. Section 3.4 and Appendix B list all training settings, and Section 4.1 specifies

the benchmark, the metrics, and how every model is run. The public sources of the data are listed in Section 3.3; part of the data comes from private industrial sources.

AI USE STATEMENT

In this work, we used generative AI tools to construct and clean the dataset: GPT-5.2 reconstructs Build123d programs from STEP models and orthographic views and writes the geometric and sequence descriptions (Section 3.3 and Appendix C). Every reconstructed program is executed and compared with its source geometry, and the descriptions are checked by repeated human sampling. We also used generative AI tools to refine the framing of the research question and hypothesis, to give feedback on the design of the experiments, and to help interpret the results. Additionally, we used generative AI tools to assist in drafting and polishing parts of several sections. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Mohammadmehdi Ataei, Farzaneh Askari, Kamal Rahimi Malekshan, and Pradeep Kumar Jayaraman. Zero-to-CAD: Agentic synthesis of interpretable CAD programs at million-scale without real data. *arXiv preprint arXiv:2604.24479*, 2026.

build123d Contributors. build123d documentation. https://build123d.readthedocs.io/, 2026.

Evan Casey, Tianyu Zhang, Shu Ishida, John Roger Thompson, Amir Khasahmadi, Joseph George Lambourne, Pradeep Kumar Jayaraman, and Karl D. D. Willis. Aligning constraint generation with design intent in parametric CAD. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8613–8622, 2025.

DeepSeek-AI. DeepSeek-V4, 2026. Model release.

Xiaoyu Dong, Zhi Li, and Xiao-Ming Wu. MUSE: Benchmarking manufacturable, functional, and assemblable text-to-CAD generation. *arXiv preprint arXiv:2605.28579*, 2026a.

Xintong Dong, Chuanyang Li, Peng Zheng, Chuqi Han, Jiaxin Jing, Hailong Shen, Yanzhi Song, and Zhouwang Yang. HistCAD: A constraint-aware parametric history-based CAD representation, dataset, and benchmark with industrial complexity. *arXiv preprint arXiv:2602.19171*, 2026b.

Maksim Elistratov, Marina Barannikov, Gregory Ivanov, Valentin Khrulkov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. CADEvolve: Creating realistic CAD via program evolution. *arXiv preprint arXiv:2602.16317*, 2026.

Prashant Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. CADmium: Fine-tuning code language models for text-driven sequential CAD design. *Transactions on Machine Learning Research*, 2026.

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, pages 59765–59789, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

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*, 2024.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A big CAD model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9601–9611, 2019.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal CAD reconstruction with reinforcement learning. In *International Conference on Learning Representations*, 2026.

Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. CAD-Llama: Leveraging large language models for computer-aided design parametric 3D model generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18563–18573, 2025.

Jiahao Li, Yusheng Luo, Yunzhong Lou, and Xiangdong Zhou. ReCAD: Reinforcement learning enhanced parametric CAD model generation with vision-language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 6190–6198, 2026a.

Jiahao Li, Qingwang Zhang, Qiuyu Chen, Guozhan Qiu, Yunzhong Lou, and Xiangdong Zhou. Towards high-fidelity CAD generation via LLM-driven program generation and text-based B-Rep primitive grounding. In *ICML*, 2026b.

Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Seek-CAD: A self-refined generative modeling for 3D parametric CAD using local inference via DeepSeek. In *ICLR*, 2026c.

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.

Moonshot AI. Meet Kimi K2.6: Advancing open-source coding. https://forum.moonshot.ai/t/meet-kimi-k2-6-advancing-open-source-coding/369, 2026.

Ke Niu, Haiyang Yu, Zhuofan Chen, Mengyang Zhao, Teng Fu, Bin Li, and Xiangyang Xue. From intent to execution: Multimodal chain-of-thought reinforcement learning for precise CAD code generation. *arXiv preprint arXiv:2508.10118*, 2025.

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*, 2026.

Qwen Team. Qwen3.5: Towards native multimodal agents, 2026a. Model release blog.

Qwen Team. Qwen3.6-Max-Preview, 2026b. Model release.

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*, pages 9801–9811, 2025.

Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. *arXiv preprint arXiv:2007.08506*, 2020.

Weijia Shi, Sewon Min, Maria Lomeli, Chunting Zhou, Margaret Li, Gergely Szilvasy, Rich James, Xi Victoria Lin, Noah A. Smith, Luke Zettlemoyer, Scott Yih, and Mike Lewis. In-context pretraining: Language modeling beyond document boundaries. In *International Conference on Learning Representations*, 2024.

Harmanjot Singh, Abhra Dubey, and Jorge Alejandro Amador Herrera. CADEngBench: It looks like CAD, but does it work? evaluating parametric design, assembly reasoning, and physics simulation. *arXiv preprint arXiv:2608.09296*, 2026.

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5-6):544–557, 2009.

Liang Wang, Heng Meng, Zekai Xiang, Jin Liu, Pingyi Zhou, Litao Chen, and Yongqiang Tang. Text2CAD-Bench: A benchmark for LLM-based text-to-parametric CAD generation. *arXiv preprint arXiv:2605.18430*, 2026.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-CAD generation through infusing visual feedback in large language models. In *International Conference on Machine Learning*, 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), 2021.

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

Haoyang Xie and Feng Ju. Text-to-CadQuery: A new paradigm for CAD generation with scalable large model capabilities. *arXiv preprint arXiv:2505.06507*, 2025.

Z.ai. GLM-5.2, 2026. Model releases.

A PRE-TRAINING CORPUS

Notebooks are converted to interleaved text and code cells, Markdown images are reduced to their alt text, text is normalized to Unicode NFC, and credentials such as access tokens and private keys are redacted. Binary, auto-generated, and oversized files are skipped.

B TRAINING DETAILS

Table 4: Training configuration of the IntentCAD model. Empty cells are settings we do not report for continual pre-training.

Setting	Stage 1: pre-training	Stage 2: fine-tuning
Initialization	Qwen3.5-4B	Stage 1 checkpoint
Trainable parameters	all	LoRA on all linear layers
LoRA rank, α, dropout		32, 64, 0
Objective	next-token prediction	target program only
Learning rate	5×10^{-6}	1×10^{-4}
Schedule		cosine with linear warmup
Epochs	10	3
Optimizer		AdamW, weight decay 0
Gradient clipping		1.0
Global batch size		256 (1 per GPU, 8 accumulation)
GPUs		32
Max sequence length		8,192 tokens, no packing
Chat template		Qwen3.5, non-thinking
Precision	bf16	bf16
Seed		42

C PROMPTS

The three prompts below are used with GPT-5.2 to reconstruct programs and to write the two descriptions. The feature-tree header that the first prompt requests is removed from each program before training. Each description is returned as a JSON object with a feature tree, whose nodes have the fields `id`, `parent_id`, `role`, `geometry`, `reference`, `dimensions`, `placement`, and `topological_effect`, and the description text.

Program reconstruction.

```
You are a mechanical CAD engineer with expert knowledge of build123d. Reconstruct the target
     part as executable build123d Python code from the supplied STEP geometry data and the
     front, top, and side orthographic views.

Source-of-truth order
1. The STEP topology, measurements, solid count, and coordinate bounds are the authoritative
     geometric evidence.
2. Use the orthographic views to resolve appearance, feature direction, occlusion, and
     symmetry. If an image projection conflicts with a STEP measurement, follow the STEP
     evidence.
3. Do not infer holes, slots, fillets, chamfers, threads, internal structures, or other
     features from a part name, variable name, or general convention when the supplied
     evidence does not establish them.

Modeling requirements
1. Begin with `from build123d import *`. Produce self-contained code. Do not read or import
     the source STEP, access the network or filesystem, start subprocesses, or invoke a
     viewer.
2. Preserve the STEP coordinate system, orientation, position, and scale. Do not recenter,
     rotate, mirror, or rescale the part unless the supplied evidence explicitly requires it.
3. Use millimeters for all dimensions. Write independent dimensions with at most two decimal
     places. Derive radii, half-dimensions, offsets, and spacing from primary parameters
     when possible instead of introducing repeated approximations.
4. Define dimensions and positions as clearly named parameters near the start of the
     program. Prefer stable sketches, extrusions, revolves, lofts, Boolean operations,
     fillets, and chamfers. Avoid fragile face or edge indexing when a geometric selector or
     explicit construction plane can express the intent.
5. Use a clear modeling order: parameters, main body, additive features, subtractive
     features, then fillets or chamfers. Every operation must affect the target solid; omit
     ineffective operations and explanatory geometry that does not belong to the final part.
6. The final value must be one valid, positive-volume solid consistent with the STEP
     evidence and must be assigned to `result` at the end of the script. Do not call `show`,
     `show_object`, `display`, or any export function.
7. The code must run without manual editing. Include only short comments that clarify major
     modeling stages.
8. Construct one global topology feature tree for the complete part before writing the
     operations. The root represents the final solid. Its ordered nodes must cover the base
     body, additive features, subtractive features, repeated or mirrored feature groups, and
     finishing features. Each node must state a stable feature ID, parent ID, role,
     construction plane or axis, defining dimensions and placement, and its topological
     effect on the parent. This is a valid reconstruction tree aligned with the generated
     code, not a claim about unavailable design history.

Output contract
Return only plain Python source code. At the beginning of the file, after the import,
     include the global topology feature tree as concise `#` comments using ASCII
     indentation and the exact feature IDs used by the subsequent stage comments. Then
     provide the executable model. Do not return Markdown fences, JSON, explanations,
     reasoning, greetings, or any text outside the code. Preserve the final newline.
```

Geometric description (Geo).

```
You are a mechanical CAD geometry specialist. Using the supplied STEP geometry data, front,
     top, and side orthographic views, and final build123d code, write a precise and
     complete description of the final geometry.

Source-of-truth order
1. The STEP topology, measurements, solid count, and coordinate bounds are the authoritative
     geometric evidence.
2. Use the final code to identify parameter meaning, feature placement, and the actual
     result of each Boolean operation.
3. Use the orthographic views to resolve appearance, direction, occlusion, and symmetry. If
     a view conflicts with the STEP evidence, follow the STEP evidence.
4. Do not invent features or uses from names, variable names, or general convention.
```

```
Description requirements
1. Describe only the final effective geometry. Do not narrate modeling operations, and do
     not mention the STEP file, views, code, input, evidence, or reasoning process.
2. Proceed from the overall form to local details. State the neutral part type, overall
     dimensions, main body shape, and all actual holes, slots, bosses, pockets, ribs,
     fillets, chamfers, and related features.
3. Include every dimension, count, axis, position, spacing, depth, through or blind
     condition, intersection, containment, pattern, mirror, and symmetry relation needed to
     determine the geometry.
4. Distinguish diameter from radius, actual solid height from tool travel, overall length
     from center-to-center distance, and through holes from blind holes. Do not describe an
     ineffective Boolean operation as a feature of the final solid.
5. Use millimeters. Numeric literals may have at most two decimal places. Express derived
     dimensions with diameters, complete dimensions, coordinate ranges, center distances, or
     symmetry relations when that avoids inaccurate half-dimension approximations.
6. Produce one global topology feature tree for the complete final part. The root represents
     the final solid. Ordered nodes must cover the base body, additive features, subtractive
     features, repeated or mirrored feature groups, and finishing features. Every node must
     contain `id`, `parent_id`, `role`, `geometry`, `reference`, `dimensions`, `placement`,
     and `topological_effect`. Group repeated identical features in one node and state their
     count and distribution. The tree must describe a valid reconstruction supported by the
     final geometry and code, not assert unavailable design history.
7. The first paragraph must be one short sentence identifying a neutral part category and
     only a use directly supported by the evidence. If the use is uncertain, state only the
     part category.
8. The second paragraph must state the design intent supported by the geometry: load path,
     support, mounting, alignment, clearance, motion, sealing, symmetry, manufacturability,
     or interface behavior only when the corresponding feature relationships establish it.
     When application intent is uncertain, describe only geometric intent, such as
     maintaining symmetry, locating mating geometry, providing clearance, or distributing
     repeated interfaces. Do not invent an industry, assembly, material, manufacturing
     process, load value, tolerance, or operating condition.
9. The third paragraph must contain the complete geometric description from the overall body
     to local details.
10. Use natural, precise, unambiguous prose. Do not use headings, lists, code, Markdown,
     commentary, or evaluative language inside `description`.

Output contract
Return exactly one JSON object with only `topology_feature_tree` and `description`.
     `topology_feature_tree` must be an array of node objects using the required fields,
     with the root first and dependencies before dependents. `description` must contain
     exactly three prose paragraphs separated by one blank line: category or supported use,
     design intent, and final geometry. Return no other fields or explanation.
```

Sequence description (Seq).

```
You are a mechanical CAD modeling specialist. Using the supplied STEP geometry data, front,
     top, and side orthographic views, and final build123d code, write a precise and
     complete description of the effective modeling sequence.

Source-of-truth order
1. The final build123d code determines the effective operations and their order.
2. Use the STEP topology, measurements, solid count, and coordinate bounds to verify the
     actual geometric result of each operation.
3. Use the orthographic views to resolve feature direction, appearance, occlusion, and
     symmetry. If a view conflicts with the STEP evidence, follow the STEP evidence.
4. Do not invent operations or features from names, variable names, or general convention
     when they are not implemented by the final code.

Description requirements
1. Follow the effective operation order in the final code. Do not reduce the answer to an
     appearance summary, and do not mention the STEP file, views, code, input, evidence, or
     reasoning process.
2. For each stage, state the sketch or reference plane, profile type, required dimensions,
     center or placement method, operation direction, distance, Boolean mode, and actual
     resulting feature.
3. Explain the dependencies among extrusions, revolves, lofts, mirrors, patterns, cuts,
     fillets, and chamfers. For repeated features, state their count, distribution, spacing,
     and symmetry relation.
4. Distinguish diameter from radius, actual solid height from tool travel, overall length
     from center-to-center distance, and through holes from blind holes. Do not claim that
     an operation creates a feature when it does not alter the solid; mention such an
     operation only when necessary and explicitly state that it leaves the final solid
     unchanged.
5. Use millimeters. Numeric literals may have at most two decimal places. Express derived
     dimensions through primary parameter relations, complete dimensions, coordinate ranges,
     center distances, or symmetry when that avoids inaccurate approximations.
```

```
6. Produce one global topology feature tree aligned with the effective code sequence. The
     root represents the final solid. Ordered nodes must cover the base body, additive
     features, subtractive features, repeated or mirrored feature groups, and finishing
     features. Every node must contain `id`, `parent_id`, `role`, `geometry`, `reference`,
     `dimensions`, `placement`, and `topological_effect`. Group repeated identical features
     in one node and state their count and distribution. Dependencies must precede their
     dependents, and node IDs must be used consistently in the sequence description.
7. The first paragraph must be one short sentence identifying a neutral part category and
     only a use directly supported by the evidence. If the use is uncertain, state only the
     part category.
8. The second paragraph must state the design intent supported by the effective feature
     relationships: load path, support, mounting, alignment, clearance, motion, sealing,
     symmetry, manufacturability, or interface behavior only when those relationships
     establish it. When application intent is uncertain, describe only geometric intent. Do
     not invent an industry, assembly, material, manufacturing process, load value,
     tolerance, or operating condition.
9. The third paragraph must contain the complete modeling sequence and cite the
     corresponding feature IDs from `topology_feature_tree` as plain parenthetical
     references.
10. Use natural, precise, unambiguous prose. Do not use headings, lists, code, Markdown,
     commentary, or evaluative language inside `description`.

Output contract
Return exactly one JSON object with only `topology_feature_tree` and `description`.
     `topology_feature_tree` must be an array of node objects using the required fields,
     with the root first and dependencies before dependents. `description` must contain
     exactly three prose paragraphs separated by one blank line: category or supported use,
     design intent, and effective modeling sequence. Return no other fields or explanation.
```