CAD-Coder:Text-Guided CAD Files Code Generation

Changqi He
hechangqi235@outlook.com

Shuhan Zhang
tigerzh7@hrbeu.edu.cn

Liguo Zhang
zhangliguo@hrbeu.edu.cn

Jiajun Miao
miaojiajun@hrbeu.edu.cn

College of Computer Science and Technology, Harbin Engineering University, China

Abstract

Computer-aided design (CAD) is a way to digitally create 2D drawings and 3D models of real-world products. Traditional CAD typically relies on hand-drawing by experts or modifications of existing library files, which doesn't allow for rapid personalization. With the emergence of generative artificial intelligence, convenient and efficient personalized CAD generation has become possible. However, existing generative methods typically produce outputs that lack interactive editability and geometric annotations, limiting their practical applications in manufacturing. To enable interactive generative CAD, we propose CAD-Coder, a framework that transforms natural language instructions into CAD script codes, which can be executed in Python environments to generate human-editable CAD files (.Dxf). To facilitate the generation of editable CAD sketches with annotation information, we construct a comprehensive dataset comprising 29,130 Dxf files with their corresponding script codes, where each sketch preserves both editability and geometric annotations. We evaluate CAD-Coder on various 2D/3D CAD generation tasks against existing methods, demonstrating superior interactive capabilities while uniquely providing editable sketches with geometric annotations.

1. Introduction

Generative AI is profoundly reshaping the production mode and innovation path in industrial field, and promoting the intelligent transformation of traditional methods in machinery,construction, automobile and other fields. Computer-aided design (CAD) holds significant importance in industrial design. However, traditional CAD models require manual drafting by professionals, which demands a certain level of expertise from engineers, and is often inefficient and prone to errors. In recent years, with the advancement of generative AI and 3D generation, research related
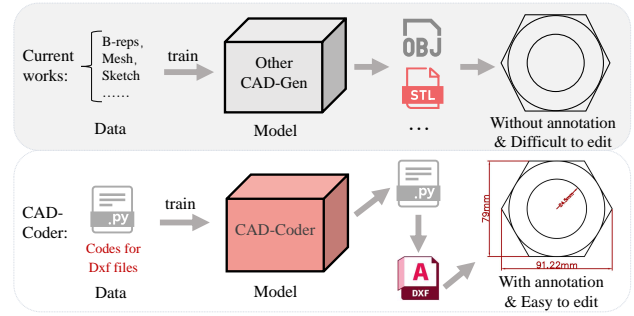

Figure 1. **Comparison of CAD-Coder with current works.** Comparing with other CAD generation methods, CAD-Coder uses a different form of dataset and produces more easily editable and annotated CAD models.

to CAD generation has garnered widespread attention.

Early research primarily focused on reconstructing CAD models from 3D point clouds[12, 14, 27, 45], or generating CAD models based on CAD command sequences[28, 38, 40]. However, due to the high complexity of the above data format, these methods cannot guide the generation of CAD models through natural language, resulting in their poor practicability. Consequently, recent scholarly efforts have shifted towards text-guided CAD model generation [13, 22, 24, 41], enabling users to obtain desired CAD models through natural language descriptions. Despite these advances, the outputs of these models fail to produce underlying universal CAD format files, making them still challenging for engineers to utilize in practical applications as shown in Figure 1.

Inspired by large language models(LLMs) such as Chat-GPT [11], Llama [18, 35, 36] and code generation tasks like CodeLlama[30] , Codex[16], CodeGeeX[46], Phi[10], etc., we propose CAD-Coder, a Python-based network for generating CAD sketches and models, as shown in Figure 2. By writing Python script code with the ezdxf library, we construct easily editable underlying universal CAD format files

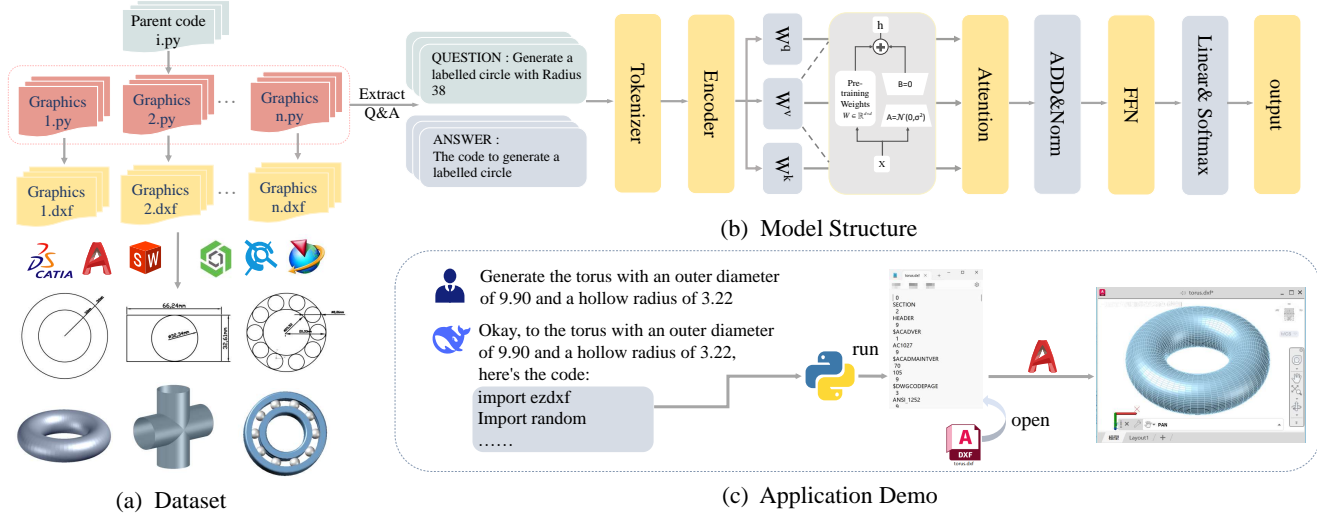(a) Dataset   (b) Model Structure   (c) Application Demo

Figure 2. **Pipeline of the CAD-Coder.** By randomly assigning values to the free parameters in the parent code, a series of script codes along with their corresponding Dxf files are generated, forming the CFSC Dataset. The dataset contains both 3D models and 2D sketches, especially contains annotated data. The codes in dataset are matched with their corresponding natural language descriptions, from which relevant question-answer pairs were extracted and injected into the DeepSeek-R1-Distill-Llama-8B model. Through training with the LoRA method, the model acquired the ability to infer CAD script code. Users can query the model to obtain the desired CAD script code, which can then be executed to generate Dxf files. These files can be opened and edited on various platforms.

(.Dxf) and achieve cross-modal generation from natural language to script code, and ultimately to CAD model. The resulting Dxf files of the CAD models [34] can be directly opened by common CAD platforms such as AutoCAD[2], SolidWorks[7], CAXA[3], CATIA[4], UG[8], Onshape[6] and others.

Designing components and products requires precise geometric design and structural analysis. For practical CAD models, data annotation[29] such as length, radius, angle, tolerance, chamfer and surface roughness are indispensable. However, current models are largely incapable of generating CAD models with annotation information, primarily due to limitations in existing datasets. To address this, we construct a dataset, CAD Files Script Code (CFSC) Dataset, which leverages Python's ezdxf library[9] to construct CAD models. This dataset consists of Python script codes is capable of constructing CAD files alongside their corresponding CAD file outputs. To sum up, the main contributions of this work include:

- We propose CAD-Coder, a CAD generation model capable of producing easily editable CAD models based on textual input.
- Leveraging the annotation features of the ezdxf library for CAD sketches, our model enables accurate data annotation for 2D sketches.
- We introduce the CFSC dataset, which includes a large number of CAD models along with their corresponding script codes and natural language descriptions.

2. Related Work

CAD generative model. Early work on sketch generation primarily focused on 2D sketch generation. Models such as SketchGen[28] and CAD-as-language[20] utilized the Transformer[37] architecture to handle geometric constraints in 2D sketches for generating 2D sketches. Later, models like DeepCAD[40], SkexGen[42], and Draw Step by Step[26] advanced the field by outputting 3D CAD operation sequences based on the Transformer architecture, marking a significant step toward generative CAD in the 3D domain. Additionally, the BrepGen[43] model directly generated 3D models in B-rep format through structured latent geometric representations. However, none of these models possess the capability to generate CAD sketches or models based on natural language. The latest advancements in CAD generation research, such as Text2CAD[22] and LLM4CAD[24], leveraged LLMs to achieve text-guided CAD generation. However, these approaches are constrained by dataset limitations, lacking the capability to produce diverse models. Additionally, they are unable to generate annotated sketches.

CAD sketch and program synthesis. A CAD sketch typically consists of one or more closed graphs (loops), with each loop composed of multiple primitive geometric elements such as lines, arcs, circles and so on. Designers can assist in sketch design through commands or programming languages[21, 33]. Among existing methods, AutoLisp[1] and FreeCAD[5] commands are relatively pop-

ular. Autolisp[1], as the built-in scripting language for AutoCAD[2], offers strong interactivity but is limited in cross-platform applications. The FreeCAD[5] command interface is open-source and excels in geometric processing capabilities, yet its efficiency in handling complex Dxf files needs improvement[34]. Both methods are constrained by platform limitations, lacking strong universality and requiring a high level of expertise.

CAD dataset. Existing CAD datasets can be categorized into two types: 2D sketches and 3D models. In the realm of 2D sketches, datasets such as SketchGraphs[31], Vitruvion[32] and CAD-as-language[20] have established structured representation Fusion 360 Gallery[39], ABC[23], Thingi10K[47], and CC3D[17, 19] accomplish model generation tasks through CAD construction sequences or B-Rep formats. However, these datasets generally lack paired text descriptions and CAD models, leading to limitations in text-driven generation tasks. Although Text2CAD[22] pioneered the construction of a cross-modal dataset linking text prompts with CAD command sequences, existing datasets still share a common flaw: 2D sketches lack explicit geometric annotations.

3. Methodology

This section details the framework design methodology of the proposed CAD-Coder. We divide this section into three parts, including the core framework of CAD-Coder for CAD model generation (Sec. 3.1), the construction process of our CFSC Dataset (Sec. 3.2) and the elaboration on the training strategy of CAD-Coder in detail(Sec. 3.3).

3.1. CAD underlying universal file

As mentioned above, current CAD generation models are unable to produce editable CAD models. To address this issue, we use Dxf files as the final output format for CAD-Coder. Dxf file is a underlying universal format for Computer-aided design, primarily used to store and exchange 2D or 3D design data, known for their high precision and editability[34]. However, since Dxf files, result in lengthy text formats that are complex and cumbersome after parsed, they are challenging to generate directly. Consequently, CAD-Coder leverages Python script codes to generate Dxf files, which is also adapted to the working principle of LLMs.

3.2. CAD Files Script Code Dataset

To better train CAD-Coder for generating annotated CAD models, we introduce the CFSC Dataset, which comprises 29,130 Dxf files of CAD models along with their corresponding script codes. The dataset includes 2D sketches without annotations, 2D sketches with annotations, and 3D models. The statistical details of geometric primitives and data annotations in the dataset are presented in the Figure 3.
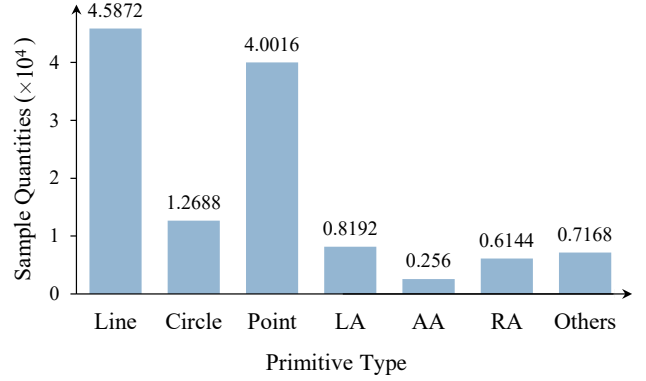

Figure 3. **Quantities of Different Primitive Types.** LA stands for linear annotation, AA is angle annotation, and RA is radius annotation.

Specifically, for a given shape, we first develop a framework script code **P** that incorporates all the necessary constraint relationships for the shape. However, instead of providing actual dimensional information, **P** references a set of parametric variables $v1$, $v2$, ..., vn to represent the fundamental characteristics of the shape, such as dimensions and position. Subsequently, we employ a randomization algorithm R to assign values to the parameter set $v1$, $v2$, ..., vn, thereby generating diverse shape script codes. It is
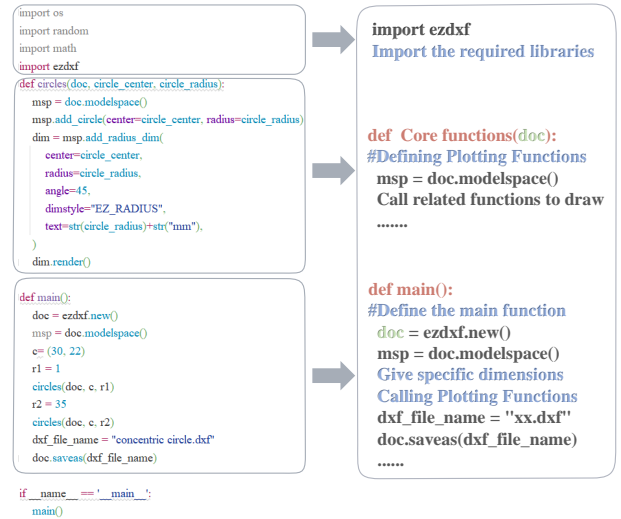

Figure 4. **Script code structure illustration.** The script code for each Dxf file consists of: (1) importing library functions, (2) the model construction function, and (3) the main function. The model construction function is responsible for defining the script code framework, including both model construction and adding annotations. The main function is used to call the model construction function, define the required dimensional parameters.

important to note that the randomization algorithm R must account for the legality of the shape. For example, in the case of a hexagonal nut, the nominal diameter D must be smaller than the short diameter De of the hexagon. Clearly, such randomization algorithms are not uniform across different shapes. We will list the constraints required for various shapes in supplementary material.

By repeatedly sampling the parameter set, we can generate a series of subscript codes $p1$, $p2$, ..., pn, each corresponding to a different combination of parameters. These subscript codes independently produce their respective Dxf files. To better adapt to the code generation task, we standardize the structure of script codes, as shown in Figure 4. The unified script code structure results in high similarity
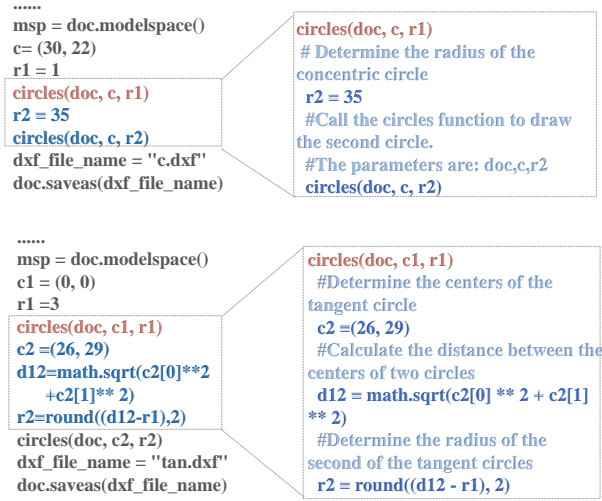

Figure 5. **Handling of similar script code segments.** Comments are added to script code segments that are prone to confusion, making it easier for the model to differentiate between these data.

in script codes for certain shapes. For example, the script codes for concentric circles and tangent circles differ only minimally in the main function, as shown in Figure 5. Such minor differences are prevalent and difficult to capture during model training, leading to instability in the model's performance when generating script codes. When the model infers "circles(doc, c, r1)", it cannot determine whether the next line should generate $r2 = 35$ or $c2 = (26, 29)$. To address this issue, we use the LLM to add comments to the script codes, particularly in areas prone to such discrepancies. In fact, annotating the parent script code **P** alone suffices to annotate an entire set of data, significantly reducing the workload.

3.3. Model Architecture

Data preprocessing. To enable the LLMs to adapt to the CAD generation task and learn the script codes patterns better in our dataset, we need to perform data preprocessing

by extracting 'Natural language description - Python script code' question-answer pairs from the dataset. Specifically, we create corresponding script codes for each CAD model and provide appropriate prompts, forming sets of question-answer pairs $\langle q_i, a_i \rangle$, where q_i is the natural language description of model i, and a_i is the script code of model i. Subsequently, we use the Byte Pair Encoding (BPE) algorithm to tokenize the data, converting the original questions and script code snippets into a series of token sequences:

$$q_i = (t_{i,1}, t_{i,2}, ..., t_{i,Lq}, c_{i,1}, c_{i,2}, ..., c_{i,La}), \quad (1)$$

where L_q and L_a represent the number of tokens for the question and answer respectively, t and c represent the tokens of the question and the answer, respectively.

Then, the tokens are converted into vectors through an embedding matrix E:

$$x_i = E(q_i) = (e(t_{i,1}), e(t_{i,2}), ..., e(t_{i,Lq}),$$
$$e(c_{i,1}), e(c_{i,1}), ..., e(c_{i,La})), \quad (2)$$

x_i is an encoded vectors that We will inject into the DeepSeek-R1-Distill-Llama-8B model to guide the generation of CAD models.

Distillation model. We select the DeepSeek-R1-Distill-Llama-8B[25] model as the base model. Compared to the standard Llama model, the distilled model transfers the reasoning capabilities of DeepSeek-R1 to a more lightweight 8B parameter model through knowledge distillation techniques, resulting in enhanced reasoning abilities. The primary method of distillation involves minimizing the Kullback-Leibler Divergence (KL Divergence) by comparing the output distributions of the teacher model and the distilled model:

$$\mathcal{L}_{\text{distill}} = \sum_x D_{\text{KL}}(P_{\text{teacher}}(y|x) \parallel P_{\text{student}}(y|x)), \quad (3)$$

where $P_{teacher}$ is DeepSeek-R1, and $P_{student}$ is Llama-8B, x and y refer to their respective inputs and outputs.

Then we smooth the output of the teacher model by temperature scaling to avoid overfitting the hard labels of the teacher model.

$$P_{\text{teacher}}^{\tau}(y|x) = \text{softmax}\left(\frac{\log P_{\text{teacher}}(y|x)}{\tau}\right), \quad (4)$$

The temperature parameter τ softens the distribution, making it easier for the student model to learn the implicit reasoning logic of the teacher model.

After distillation, the model further optimizes the quality of generation by combining format rewards with accuracy rewards for continued reinforcement learning (RL). DeepSeek-R1-Distill-Llama-8B possesses inference performance close to that of the full version of the R1 model, better aligning user-input cue words with the model's output content.

LoRA. To avoid significant computational and storage overhead, we employed the LoRA method during the model's learning process. The LoRA method reduces the number of parameters that need to be updated through low-rank matrix decomposition. Specifically, When injecting LoRA into the self-attention layer of the Transformer, the LoRA parameters are:

$$W' = W_0 + \Delta W = W_0 + BA, \quad (5)$$

where $W \in \mathbb{R}^{d \times k}$ is original weight, $\Delta W \in \mathbb{R}^{d \times k}$ is a low-rank matrix, $A \in \mathbb{R}^{r \times k}$, $B \in \mathbb{R}^{d \times r}$, r is the low-rank dimension, which is usually much smaller than d and k. During the fine-tuning process, only the low-rank matrices A and B need to be updated, while the original weight matrices remain unchanged.

4. Experiment

In this section, we first propose four novel metrics for measurement (Sec. 4.1). Then we conduct qualitative and quantitative evaluations of the CAD-Coder from three aspects, including script codes, 2D sketches without annotations, 2D sketchs with annotations and 3D models (Sec. 4.2). Additionally, we perform comparative experiments between CAD-Coder and existing state-of-the-art LLMs to demonstrate the comprehensive capabilities of CAD-Coder (Sec. 4.3). Furthermore, we highlight the model's exceptional performance in cross-platform compatibility (Sec. 4.4). More experiment details can be found in the supplementary material.

4.1. Metrics

To better evaluate the performance of different CAD models, we employ the following evaluation metrics in the experiments:

Function accuracy (ACC-F). Compare the generated script code with the ground truth, if the set of functions composed of all functions in the generated script code is equal to the set of functions composed of all functions in the ground truth, then it is defined as the same function of the generated script code, and the overall generative framework of the model can be judged by testing the rate of the same function of all script codes.

$$ACC - F = \frac{\sum_{j=1}^{N_c} \sum_{i=1}^{N_j} I\left[f_i = \hat{f}_i\right]}{\sum_{j=1}^{N_c} N_j}, \quad (6)$$

N_c denotes that there are N_c sets of data, N_j denotes the number of functions contained in each set of data, f_i and $\hat{f}_i$ is the set of functions in the ground truth and generated script code, respectively. I is the indicator function, which takes the value 1 when the function names are the same.

Parameter accuracy (ACC-P). Compare the parameter number of each correctly generated function in the generated script code with the ground truth to get the parameter

correctness of the generated script code, thus judging the accuracy of the details of the generated sketches and models.

$$ACC - P = \frac{\sum_{p=1}^{N_e} \sum_{i=1}^{N_e} I[p_i = \hat{p}_i]}{\sum_{p=1}^{N_e} N_p}, \quad (7)$$

N_a indicates that there is N_c function, N_p indicates the number of parameters contained in each function, p_i and $\hat{p}_i$ are the parameters of the function in the ground truth and generated script code, respectively.

Graphic accuracy (ACC-G). It is obviously not enough to judge the accuracy of the script code, we need to check whether the image is standard or not, we define ACC-G to evaluate whether the generated image is accurate or not.

$$ACC - G = \frac{\sum_{i=1}^{D} I_{\text{the graph is correct}}(x_i)}{D}, \quad (8)$$

where D denotes the number of all Dxf file script codes that can be compiled, and I is an indicator function that takes the value 1 when the generated graph xi is correct.

Annotation accuracy (ACC-A). In order to test the model's annotation ability, we define the annotation accuracy ACC-A.

$$ACC - A = \frac{\sum_{i=1}^{D} I_{\text{the annotation is correct}}(x_i)}{D}, \quad (9)$$

where the indicator function I takes the value 1 when the generated graph x_i is annotated correctly.

4.2. Experiment on Generative Ability

2D sketches without annotations. Figure 6 demonstrates CAD-Coder's capability in generating abstract images without annotations, compared with VQ-CAD[38]. As shown in the Figure 6, CAD-Coder can generally produce images that are roughly similar to the ground truth, and its generation performaance is superior to that of VQ-CAD.

2D sketches with annotations. Figure 7 demonstrates CAD-Coder's capability in generating annotated CAD models. While some minor details may not perfectly match the ground truth, the model largely reproduces the main content of the ground truth CAD sketches, and the annotations are relatively accurate.

Figure 8 demonstrates the variety of annotation types our model can handle, including tolerances, surface finishes, chamfers, angles, and more. We also evaluate the accuracy of the model's annotations. As shown in Figure 7, for simple annotation types such as linear annotations and radius annotations, our model performs well and rarely makes errors in annotation types. However, for angle annotations, a small number of results mistakenly annotate diameters as radius, leading to a higher annotation data error.

For more complex annotations, such as chamfers, tolerances, and surface roughness, the CAD-Coder's sensitivity
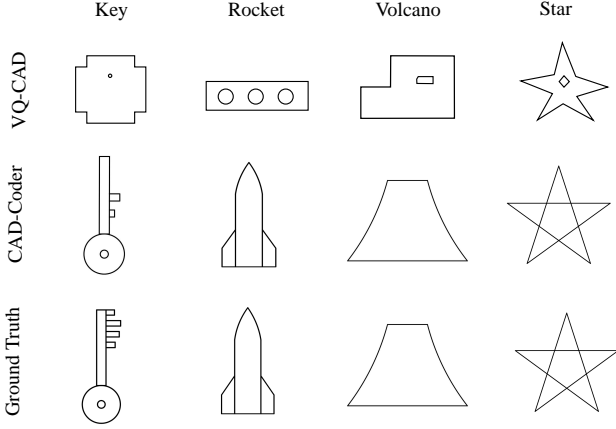
Figure 6. **Comparative evaluation of CAD-Coder and VQ-CAD.** It includes four kinds of contents: key, rocket, volcano and pentagon. In the figure, The first row is the ground truth, the second row is the result of VQ-CAD, and the third row is our result.
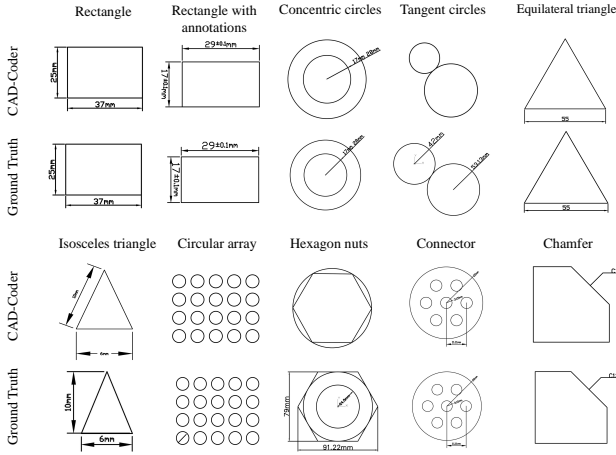

Figure 7. **Comparison of annotated sketches generated by CAD-Coder with ground truth.** There are 10 sets of comparative experimental results, and the results generated by CAD-Coder correspond to the ground truth.
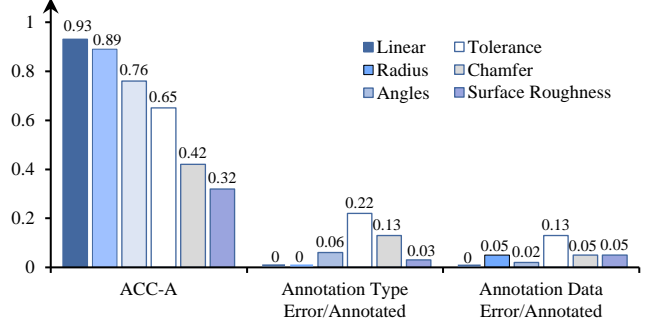

Figure 8. **Annotation capability assessment.** The bar chart includes three evaluation metrics: ACC-A (the probability of successful annotation), Annotation Type Error/Annotated (the percentage of correct annotation types in the annotation) and Annotation Data Error/Annotated (the percentage of correct data in the annotation) were tested on six annotation types, including linear annotation, radius annotation, angle annotation, tolerance annotation, chamfer annotation and surface roughness annotation.
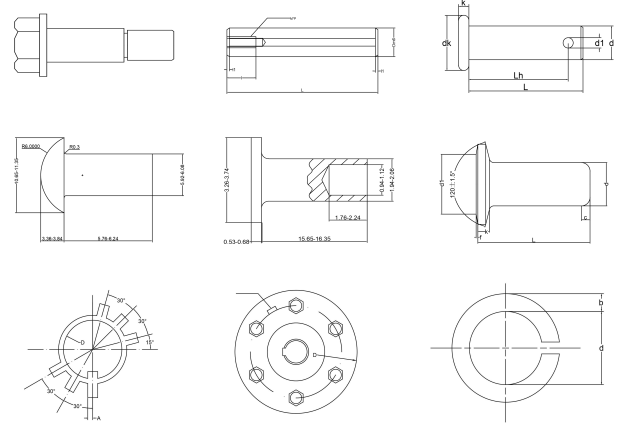

Figure 9. **More complex sketch generation.** The actual engineering parts containing various primitives and annotations are shown in this figure

is slightly lower, but it still exhibits some capability. Tolerance annotations are prone to omissions or only annotating linear dimensions, resulting in a higher annotation type error. On the other hand, chamfers and surface roughness annotations, once applied, are generally accurate in type, demonstrating a certain level of stability.

During the experiments, we observed that as the dataset and model scale increase, our method is capable of generating more complex CAD sketches, as illustrated in Figure 9. **3D Models.** Although the Dxf format is primarily used for the design of 2D sketches, it also possesses strong capabilities for representing 3D models. We did not overlook this feature of Dxf files and created a substantial number of 3D models that can be opened with Dxf files, along with

their corresponding code. After training, CAD-Coder was endowed with the ability to express 3D representations, as shown in Figure 10.

4.3. Model Comparison and Analysis

To demonstrate the capability of our method in CAD script code generation tasks, we conduct extensive generation experiments and compared our results with existing state-of-the-art models. The experiments randomly selected 485 prompts, including 212 prompts for 3D models, 115 prompts for 2D sketches without annotations, and 158 prompts for 2D sketches with annotations. The results are shown in Table 1.

Our method significantly outperforms other models across multiple metrics. The advantage in ACC-G demon-

Model	pass@1↑	pass@3↑	pass@5↑	APR↑	ACC-G↑	ACC-A ↑
Qwen2.5-Coder-14b[44]	0.14	0.26	0.33	0.39	0.16	0.09
ChatGPT-4[11]	0.16	0.33	0.39	0.49	0.17	0.03
Deepseek-V3[25]	0.13	0.17	0.18	0.54	0.25	0.29
Llama3.3-70b[18]	0.17	0.23	0.24	**0.82**	0.43	0.14
CAD-Coder	**0.40**	**0.74**	**0.81**	0.79	**0.68**	**0.77**

Table 1. **Quantitative comparison with existing LLMs.** The evaluation metrics include Pass@k[15], Average Parsing Rate (APR), Graphic Accuracy (ACC-G), and Annotation Accuracy (ACC-A). The best-performing model is highlighted in bold.

Method	ACC-F↑	ACC-P↑	ACC-G↑	pass@1↑	pass@3↑	pass@5↑	CD↓
CAD-Coder w/o. annotation	**0.79**	**0.83**	**0.69**	**0.42**	**0.79**	**0.89**	**0.74**
CAD-Coder w. annotation	0.66	0.76	0.51	0.33	0.59	0.74	0.88

Table 2. **Comparison of annotated and non-annotated CAD-Coder generation ability.** The evaluation metrics include Pass@k[15], Function Accuracy (ACC-F), Parameter Accuracy (ACC-P), Graphic Accuracy (ACC-G) and Chamfer Distance (CD).
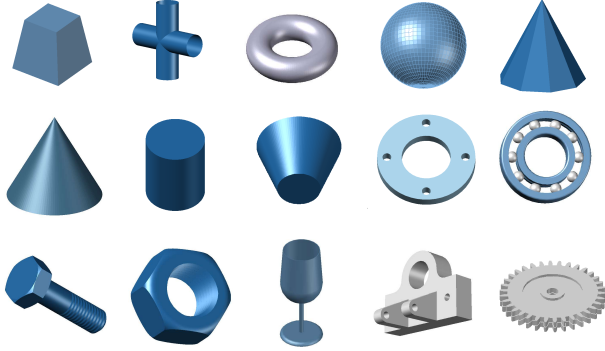

Figure 10. **Generated 3D models illustration.** CAD-Coder has a strong ability to generate 3D engineering parts including flanges, bearings, screws, nuts, gears, etc.
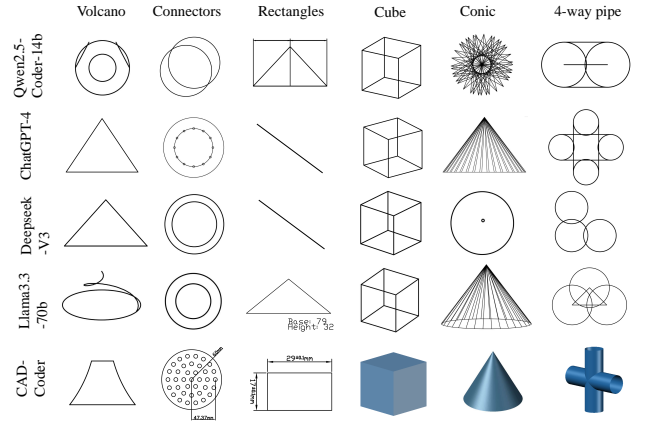

Figure 11. **Comparison of CAD-Coder's generation results with those of other LLMs.** A total of six sets of prompts are fed into the model: one set of unannotated 2D sketch prompt, two sets of annotated 2D sketch prompts, and three sets of 3D model prompts.

strates that our approach effectively translates natural language into geometric shapes, while the substantial lead in ACC-L proves that training with script code data for annotation generation is highly effective.

The Pass@k results indicate that the overall performance of our generated script codes are superior to that of other models. In terms of APR, the results of Llama are slightly better than our results, primarily because our method often generates complex but standard script codes, which have higher probability of failing to compile or run. In contrast, Llama tends to generate script codes that could compile and run but may not meet the required standards.

Figure 11 compares the CAD sketches generated by several models. The comparison clearly shows that our model generates CAD sketches with more accurate shapes, better understanding of prompts, and far superior annotation capabilities compared to other LLMs. Additionally, our model can generate more realistic 3D entities.

4.4. Experiment on Cross-Platform Capability

Existing program-driven CAD generation models typically use custom CAD commands as their output, which cannot be directly opened by common CAD platforms (e.g., AutoCAD[2], SolidWorks[7], etc.). These models still require specific tools or scripts to process the command sequences to produce visual CAD results, resulting in limited cross-platform compatibility.

Our model, however, starts with Python script codes. By running the generated Python script codes, it produces a universal underlying CAD file in Dxf format. The resulting Dxf files exhibit excellent cross-platform compatibility and can be opened in almost all mainstream CAD software and platforms, yielding the expected sketches and models, as shown in Figure12.

Model	pass@1↑	pass@3↑	pass@5↑	APR↑	ACC-G↑	ACC-P↑	ACC-F↑	CD↓
CAD-Coder w. LoRA	**0.33**	**0.59**	**0.74**	**0.79**	**0.51**	**0.76**	**0.66**	**0.88**
CAD-Coder w/o. LoRA	0.25	0.47	0.59	0.75	0.47	0.70	0.52	0.93

Table 3. **Comparison of full parameter and LoRA fine-tuning.** The evaluation metrics include Pass@k[15], Average Parsing Rate (APR), Graphic Accuracy (ACC-G), Function Accuracy (ACC-F), Parameter Accuracy (ACC-P), and Chamfer Distance (CD) The best-performing model is highlighted in bold.
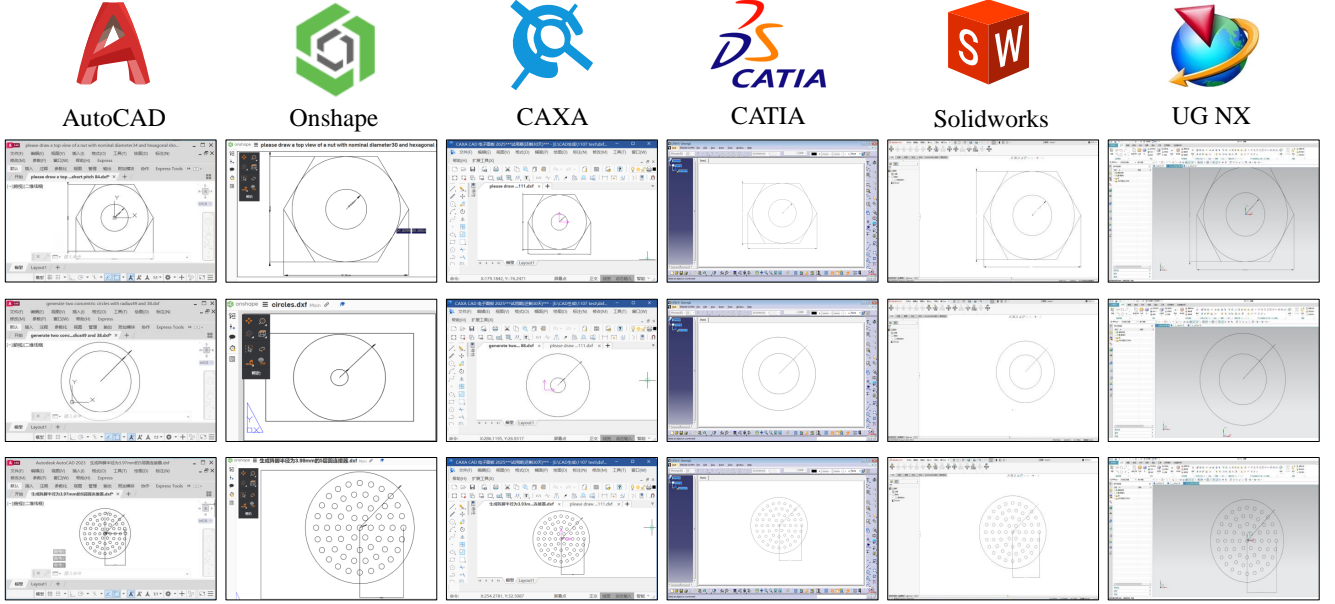

Figure 12. **Cross-platform capability illustration.** This figure displays the generated Dxf file being opened in different platforms/software, arranged from left to right as follows: AutoCAD[2], Onshape[6], CAXA[3], CATIA[4], SolidWorks[7], and UG[8].

4.5. Ablation Study

In this section, we conduct ablation studies to demonstrate the effectiveness of the proposed CAD-Coder. More ablation experiments are given in the supplementary materials.

Different Annotation Strategies. To validate our model's unique capability in generating annotated CAD sketches, we conduct ablation studies comparing performance with and without annotation generation. By leveraging the ezdxf library's annotation features, CAD-Coder pioneers automated geometric annotation in 2D CAD generation. From Table 2, we can get that the generation without annotation is better than the generation with annotated content, both for the evaluation of the script codes alone and for the evaluation of the specific generated graphs. The performance gap mainly stems from the increased complexity in annotated scripts, which require handling additional geometric relationships and parameters. This highlights both the challenge and importance of accurate annotation generation, pointing to promising directions for future improvements.

Effect of Fine-tuning Strategies. Then, we conducted full parameter fine-tuning and LoRA fine-tuning on the distilled model to compare the effects of different methods on the model's generation capabilities. From Table 3, we can get that CAD-Coder with LoRA exhibit better performance in both the accuracy of script code generation and the accuracy of CAD model generation. The results indicate that the LoRA method is more suitable for our task, as our dataset is not very large, and the LoRA method helps reduce the occurrence of overfitting.The LoRA method consumes less time, storage, and resources to complete CAD model generation tasks, making it more convenient to train a user-friendly interactive CAD generation model.

5. Conclusion

In this paper, we introduced CAD-Coder, the first interactive model capable of generating annotated CAD files from natural language descriptions. We constructed the CFSC dataset, containing 29,130 Dxf files with corresponding script codes. **For anonymous reason, this dataset will be released upon acceptance of the paper.** In the future, we plan to expand the CFSC to encompass a more diverse range of engineering components and annotation types.

References

[1] Autolisp. https://help.autodesk.com/view/OARX/2023/ENU/?guid=GUID-265AADB3-FB89-4D34-AA9D-6ADF70FF7D4B, . 2, 3

[2] Autocad. https://www.autodesk.com.cn/products/autocad/overview?term=1-YEAR&tab=subscription, . 2, 3, 7, 8

[3] Caxa. https://www.caxa.com/cad/index.html. 2, 8

[4] Catia. https://www.3ds.com/zh-hans/products/catia/all-products. 2, 8

[5] Freecad. https://www.freecad.org. 2, 3

[6] Onshape. https://www.onshape.com/en/. 2, 8

[7] Solidworks. https://www.solidworks.com/zh-hans/lp/proven-solution-3d-design-and-product-development. 2, 7, 8

[8] Ug. https://plm.sw.siemens.com/en-US/nx/cad-online/. 2, 8

[9] ezdxf. https://ezdxf.readthedocs.io/en/stable/index.html. 2

[10] Marah Abdin, Jyoti Aneja, Hany Awadalla, et al. Phi-3 technical report: A highly capable language model locally on your phone, 2024. 1

[11] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023. 1, 7

[12] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018. 1

[13] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating cad code with vision-language models for 3d designs. *arXiv preprint arXiv:2410.05340*, 2024. 1

[14] Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. Learning gradient fields for shape generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 364–381. Springer, 2020. 1

[15] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating Large Language Models Trained on Code. *arXiv e-prints*, art. arXiv:2107.03374, 2021. 7, 8, 12

[16] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code, 2021. 1

[17] Kseniya Cherenkova, Djamila Aouada, and Gleb Gusev. Pvdeconv: Point-voxel deconvolution for autoencoding cad construction in 3d. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2741–2745. IEEE, 2020. 3

[18] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 1, 7

[19] Elona Dupont, Kseniya Cherenkova, Anis Kacem, Sk Aziz Ali, Ilya Arzhannikov, Gleb Gusev, and Djamila Aouada. Cadops-net: Jointly learning cad operation types and steps from boundary-representations. In *2022 International Conference on 3D Vision (3DV)*, pages 114–123. IEEE, 2022. 3

[20] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. *Advances in Neural Information Processing Systems*, 34:5885–5897, 2021. 2, 3

[21] Shuming Gao and Jami J Shah. Automatic recognition of interacting machining features based on minimal condition subgraph. *Computer-Aided Design*, 30(9):727–739, 1998. 2

[22] Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 37:7552–7579, 2025. 1, 2, 3

[23] 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. 3

[24] Xingang Li, Yuewan Sun, and Zhenghui Sha. Llm4cad: Multimodal large language models for three-dimensional computer-aided design generation. *Journal of Computing and Information Science in Engineering*, 25(2), 2025. 1, 2

[25] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. 4, 7

[26] Weijian Ma, Shuaiqi Chen, Yunzhong Lou, Xueyang Li, and Xiangdong Zhou. Draw step by step: Reconstructing cad construction sequences from point clouds via multimodal diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 27154–27163, 2024. 2

[27] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575*, 2019. 1

[28] Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas J Guibas, and Peter Wonka. Sketchgen: Generating constrained cad sketches. *Advances in Neural Information Processing Systems*, 34:5077–5088, 2021. 1, 2

[29] B Ramani, SH Cheraghi, and JM Twomey. Cad-based integrated tolerancing system. *International journal of production research*, 36(10):2891–2910, 1998. 2

[30] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. 1

[31] 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. 3

[32] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric cad sketches. *arXiv preprint arXiv:2109.14124*, 2021. 3

[33] Jami J Shah and Martti Mäntylä. *Parametric and feature-based CAD/CAM: concepts, techniques, and applications*. John Wiley & Sons, 1995. 2

[34] Wen Shang, Jun Zhong, and Qin Yan. Analysis of dxf file with an application to 3d graphic display. In *2012 IEEE International Conference on Information and Automation*, pages 611–615. IEEE, 2012. 2, 3

[35] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1

[36] Hugo Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open foundation and fine-tuned chat models, 2023. 1

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 2

[38] Hanxiao Wang, Mingyang Zhao, Yiqun Wang, Weize Quan, and Dong-Ming Yan. Vq-cad: Computer-aided design model generation with vector quantized diffusion. *Computer Aided Geometric Design*, 111:102327, 2024. 1, 5

[39] Karl DD 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 (TOG)*, 40(4): 1–24, 2021. 3

[40] 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. 1, 2

[41] Sifan Wu, Amir Khasahmadi, Mor Katz, Pradeep Kumar Jayaraman, Yewen Pu, Karl Willis, and Bang Liu. Cad-llm: Large language model for cad generation. In *Proceedings of the neural information processing systems conference. neurIPS*, 2023. 1

[42] Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. *arXiv preprint arXiv:2207.04632*, 2022. 2

[43] Xiang Xu, Joseph Lambourne, Pradeep Jayaraman, Zhengqing Wang, Karl Willis, and Yasutaka Furukawa. Brepgen: A b-rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics (TOG)*, 43 (4):1–14, 2024. 2

[44] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. 7

[45] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019. 1

[46] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x, 2024. 1

[47] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models, 2016. 3

Supplemental Materials

The content of this supplementary material involves:

A. Experimental Setup and Costs

We performed multi-round fine-tuning of the DeepSeek-R1-Distill-Llama-8B model on a dataset containing 29130 samples, approximately 2.27 million tokens in size, using a single NVIDIA V100 GPU. The learning rate was set to 0.0002, the batch size was set to 4, the sequence length was set to 1048, and the training was conducted for 2 epochs.

B. Details of LoRA Fine-Tuning

Specifically, for the L-th layer transformer, the LoRA increment for Query/Value is:

$$Q = H_{l-1}(W_q + B_q A_q)^T = H_{l-1}W_q^T + H_{l-1}A_q^T B_q^T \tag{10}$$

Similarly, the value projection is:

$$V = H_{l-1}(W_v + B_v A_v)^T \tag{11}$$

where $B_q^{(l)}, B_v^{(l)} \in \mathbb{R}^{d \times r}, A_q^{(l)}, A_v^{(l)} \in \mathbb{R}^{r \times d}, r \ll d$ is the LoRA rank, $\Delta W \in \mathbb{R}^{d \times d}$ is the weight increment. Attention is calculated as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \tag{12}$$

Feedforward network (FFN): activated using SwiGLU, keeping the original parameters unchanged

$$\text{FFN}(X) = \text{GeLU}(XW_1^{(l)})W_2^{(l)} \tag{13}$$

Residuals and Normalization:

$$X_{\text{out}} = \text{LayerNorm}\left(X + \text{Attention} + \text{FFN}(X)\right) \tag{14}$$

C. Evaluation Metrics

Pass@k. Common evaluation metrics for code generation models refer to the criteria used to assess performance in a given generation task. If at least one of the k candidate results generated by the model meets the predefined success criteria (such as passing tests or satisfying specific conditions), the task is considered "successful." The calculation formula is as follows:

$$pass@k := E_{problems}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{15}$$

The model generates n (where $n > k$) pieces of code for each question, and then randomly selects k pieces of code from these. If at least one of the k pieces of code passes the unit tests, it is considered successful. Here, c represents the total number of pieces of code that can pass the unit tests.

We tested the results of pass@k using python code.

In addition, we used the Comparecloud platform to convert Dxf files into point cloud format files, and tested the chamfer distance (CD) of the generated results with the help of python's open3d library.

D. More Ablation Studies

We conducted three sets of ablation experiments to assess the impact of different datasets, methods, and baseline models on the generation performance of CAD-Coder.

First, we trained the DeepSeek-Distill-Qwen model using six completely different datasets. These six datasets included: training code with comments and without comments, training code containing 3D models and without 3D models, and training code with annotated and without annotated. The results indicated that when comments were added to the training code, the model's performance metrics improved across the board, with significant increases in APR and pass@k. This demonstrates that annotating the code helps the model enhance its ability to reason about context in long texts.Then, since the functions for drawing 2D and 3D sketches using the ezdxf library are entirely different and require different import libraries, we trained the model separately on a pure 2D dataset and a dataset mixed with 3D models to evaluate the impact of dataset content on model performance. The results showed that although the model's ACC-F and ACC-P experienced a certain degree of decline after mixing in 3D data, ACC-G improved. This is because the code for 3D models in the dataset is generally more concise compared to annotated 2D sketches; as long as the code compiles successfully, it yields nearly correct results. Since ACC-G is defined as the ratio of correctly outputted graphics to the total number of

Model	pass@1↑	pass@3↑	pass@5↑	APR↑	ACC-G↑	ACC-P ↑	ACC-F ↑	CD ↓
CAD-Coder with Comments	0.40	0.74	0.81	0.79	**0.68**	**0.76**	0.66	**0.88**
CAD-Coder without Comments	0.21	0.54	0.62	0.49	0.47	0.54	0.43	1.31
CAD-Coder with 3D models	0.37	0.72	0.78	0.83	0.64	0.62	0.61	0.72
CAD-Coder without 3D models	**0.45**	**0.81**	**0.90**	**0.86**	0.43	0.75	**0.68**	0.84

Table A. **Comparison of full parameter and LoRA fine-tuning.** The evaluation metrics include Pass@k[15], Average Parsing Rate (APR), Graphic Accuracy (ACC-G), Function Accuracy (ACC-F), Parameter Accuracy (ACC-P), and Chamfer Distance (CD) The best-performing model is highlighted in bold.

CAD-Coder based on	pass@1↑	pass@3↑	pass@5↑	APR↑	ACC-G↑	ACC-P ↑	ACC-F ↑	CD ↓
Llama	0.32	0.61	0.73	0.60	0.51	0.58	0.47	1.20
CodeLlama	0.27	0.54	0,65	0.51	0.46	0.47	0.34	1.76
Qwen	0.35	0.56	0.76	0.56	0.47	0.55	0.43	1.35
Qwen-Coder	0.22	0.47	0.59	0.45	0.44	0.41	0.27	2.10
DeepSeek-R1-Distill-Llama-8B	**0.40**	**0.74**	**0.81**	**0.79**	**0.68**	**0.76**	**0.66**	**0.88**
DeepSeek-R1-Distill-Qwen-7B	0.38	0.64	0.76	0.68	0.59	0.66	0.54	0.95

Table B. **Quantitative comparison with existing LLMs.** The evaluation metrics include Pass@k[15], Average Parsing Rate (APR), Graphic Accuracy (ACC-G), and Annotation Accuracy (ACC-A). The best-performing model is highlighted in bold.

successful compilations, the inclusion of 3D data actually led to an increase in ACC-G. Therefore, we conclude that the impact of 3D data on the model is not significant, and CAD-Coder possesses generation capabilities for both 2D and 3D.Finally, we compared the performance metrics of the model on a purely unannotated dataset (including unannotated 2D sketches and 3D models) with those after incorporating an annotated dataset. The results indicated that CAD-Coder performed stably and well on the unannotated dataset. However, after adding annotations, the complexity of the training code increased, resulting in a decline in the model's performance capabilities.

In the final set of ablation experiments, we selected six large models as baseline models to test the generation performance of the CAD-Coder model under different baseline conditions. The primary baselines were the original Llama and Qwen models, followed by their respective large models focused on code generation, and finally the distilled versions of these two models. The experimental results indicated that the distilled model exhibited the best generation capabilities, followed by the baseline models, while the models specifically focused on code generation performed the worst. The reason for this is that the distilled model endows CAD-Coder with stronger reasoning abilities, whereas the models dedicated to code generation, due to certain prior knowledge, tend to generate other types of code that can disrupt the experimental results.

In the second set of ablation experiments, we conducted full parameter fine-tuning and LoRA fine-tuning on the distilled model to compare the effects of different methods on the model's generation capabilities. The results indicate

that the LoRA method is more suitable for our task, as our dataset is not very large, and the LoRA method helps reduce the occurrence of overfitting.

E. Details of Size Constraints

In this section, we present the details of the size constraints used in our method.

a. Tangent circle constraint (circumtangent circle system)

$$d_{center} = R_{major} + R_{minor}$$

R_{major}: Radius of main structure circle (base circle)
R_{minor}: Dependent circle radius (constrained circle)

b. Hexagon nut opposite side width constraint

$$S_{hex} \geq 1.5d_{nominal} + 0.2d_{internal}$$

S_{hex}: Width of opposite side of nut
$d_{nominal}$: Nominal diameter
$d_{internal}$: Inner diameter of thread

c. Flange bolt hole distribution circle diameter

$$D_{pcd} \geq D_{bore} + 2.5D_{bolt}$$

D_{bore}: Pipe aperture
D_{bolt}: Nominal diameter of bolt

d. Ball quantity and size constraints of rolling bearings

$$n_{ball} = \left\lfloor \frac{\pi(D_{outer} - D_{inner})}{2.2d_{ball}} \right\rfloor$$

D_{outer}: Inner diameter of outer ring
D_{inner}: Outer diameter of inner ring

n_{ball}: Quantity of ball
d_{ball}: Diameter of ball
e. Involute gear tooth root transition curve constraints

$$\rho_{root} \geq 0.25m_n$$

m_n: The normal modulus

F. Parent Code Example

As shown in the Figure A, it displays the parent code for generating annotated rectangle script code in our dataset.

```python
import os
import random
import math
import ezdxf
from math import pi, cos, sin

#Creating a function to draw a labeled rectangle
def rectangle(doc, p0, width, height):
    msp = doc.modelspace()
    p=[]
    p.append(p0)
    p.append((width + p0[0], p0[1]))
    p.append((width + p0[0], height + p0[1]))
    p.append((p0[0], height +p0[1]))
    msp.add_lwpolyline([p[0], p[1], p[2], p[3], p[0]], close=True)
    dim_h = msp.add_linear_dim(
        base=((width / 2) + p0[0],p0[1] - 5),
        p1=p[0],
        p2=p[2],
        dimstyle='Standard',
        text=str(str(width)+"mm"),
    ).render()

    dim_v = msp.add_linear_dim(
        base=(p0[0] - 5, (height / 2) + p0[1]),
        p1=p0,
        p2=(p0[0], height + p0[1]),
        dimstyle='Standard',
        angle=90,
        text=str(str(height)+"mm"),
    ).render()

def main(run_number):
    doc = ezdxf.new()
    msp = doc.modelspace()
    width = random.randint(10, 100)
    height = random.randint(10, 100)
    x0 = random.randint(0, 100)
    y0 = random.randint(0, 100)
    p0=[x0,y0]
    rectangle(doc,p0, width, height)
    dxf_output_dir = r"E:\10.25_t\dxf"
    py_output_dir = r"E:\10.25_t\py"
    if not os.path.exists(dxf_output_dir):
        os.makedirs(dxf_output_dir)
    if not os.path.exists(py_output_dir):
        os.makedirs(py_output_dir)
    # Save DXF file with r1 and r2 in the name
    dxf_file_name = os.path.join(dxf_output_dir,f"the width of the rectangle is
{width},the height is{height}and the origin is {(x0,y0)}.dxf")
    doc.saveas(dxf_file_name)
    # Prepare new Python code, with hardcoded random numbers
    new_code = f"""import os
import random
import math
import ezdxf
from math import pi, cos, sin
#Creating a function to draw a labeled rectangle
def rectangle(doc, p0, width, height):
```

```python
    msp = doc.modelspace()
    #Determine the four vertices of the rectangle from the base point p0
    p=[]
    p.append(p0)
    p.append((width + p0[0], p0[1]))
    p.append((width + p0[0], height + p0[1]))
    p.append((p0[0], height +p0[1]))
    # Call the msp.add_lwpolyline function to connect the four vertices of
the rectangle
    msp.add_lwpolyline([p[0], p[1], p[2], p[3], p[0]], close=True)
    # Call the msp.add_linear_dim function to add Horizontal
Dimensions
    dim_h = msp.add_linear_dim(
        base=((width / 2) + p0[0],p0[1] - 5), # Dimensional baseline position
        p1=p[0],  # First measurement point
        p2=p[2],  # Second measurement point
        dimstyle='Standard',  # Use standard size styles
        text=str(str(width)+"mm"), # Display width
    ).render()
    # Call the msp.add_linear_dim function to add vertical dimensioning
(using different settings)
    dim_v = msp.add_linear_dim(
        base=(p0[0] - 5, (height / 2) + p0[1]),  # Dimensional baseline
position
        p1=p0,  # 1st measurement point
        p2=(p0[0], height + p0[1]),  # 2nd measurement point
        dimstyle='Standard',  # Use standard size styles
        angle=90,#Give Goga a labeling line with an angle of 90
        text=str(str(height)+"mm"), #Display height
    ).render()
def main():
    doc = ezdxf.new()
    msp = doc.modelspace()
    #Given the width and height of a rectangle
    width = {width}#Rectangle width {width}mm
    height = {height}##Rectangle height {height}mm
    x0 = {x0}#Define the horizontal coordinates of the vertices of the
lower left corner of the rectangle
    y0 ={y0}#Define the vertical coordinates of the vertices of the lower
left corner of the rectangle
    p0=[x0,y0]#Define the lower left corner vertex of the rectangle
    #Calling a function that generates a labeled rectangle
    #The parameters are: doc, p0,width,height
    rectangle(doc,p0, width, height)
    #Save as a dxf file named as rectangle
    dxf_file_name = "rectangle.dxf"
    doc.saveas(dxf_file_name)
if __name__ == '__main__':
    main()
"""
    py_file_name = os.path.join(py_output_dir,f"please draw a rectangle
for me ,the width
    of the rectangle is{width},the height is{height}and the origin is
{(x0,y0)}.py")
    with open(py_file_name, "w") as f:
        f.write(new_code)
if __name__ == '__main__':
    for i in range(1, 501):
        main(i)
```

Figure A. Annotated rectangle generation parent code.