Fast Text-to-CadQuery: Scalable CAD Generation with Large Models

Shengjun Xia, Zhikang Xie, Xinran Chen, Yukai Zhou
Department of Computer Science, Fudan University
{22307110187, 22300240015}@m.fudan.edu.cn

Abstract

Computer-aided design (CAD) is a vital tool across various industries like engineering, architecture, and 3D printing, yet its complexity poses a significant barrier for non-experts. The emergence of large language models (LLMs) and multimodal models offers a promising pathway to democratize CAD interaction. Our research introduces a novel approach that leverages CadQuery's high-level API to directly generate executable 3D models from natural language descriptions, sidestepping the need for intermediate command sequences. We utilized the existing Text2CAD dataset by downloading and extensively cleaning it to ensure the executability and accuracy of the CadQuery code, resulting in a robust data resource comprising approximately 170,000 high-quality samples. Our comprehensive evaluation of the Qwen2.5-3B model reveals that larger models significantly enhance generation accuracy. Furthermore, we applied advanced training techniques, including Supervised Fine-Tuning (SFT), Low-Rank Adaptation (LoRA), and model distillation, each contributing to substantial improvements in model performance. These techniques allowed us to refine the model's capabilities, offering an optimized solution for symbolic 3D generation. Our findings underscore that directly generating CadQuery code simplifies the modeling pipeline, leveraging pretrained models' strengths in Python code generation and spatial reasoning, thereby paving the way for scalable and efficient CAD processes. Project page: `https://github.com/kurumi8686/FDU2025-Computer-Graphics/tree/main/FinalPJ`.

1 Introduction

Computer-aided design (CAD) is an indispensable cornerstone across diverse fields such as engineering, architecture, construction, and 3D printing. It empowers professionals to create precise digital models, facilitating rigorous analysis, comprehensive simulation, and efficient fabrication processes [13, 3]. However, the creation of sophisticated CAD models traditionally demands specialized technical expertise and deep familiarity with complex, domain-specific software [2]. This inherent complexity presents a significant hurdle, limiting access for non-expert users and often slowing down the iterative design cycle.

The recent emergence of powerful foundation models, including large language models (LLMs) and multimodal models, offers a transformative pathway to democratize CAD interaction. Concurrently, the research community has seen a proliferation of new datasets specifically tailored for generative CAD, exploring various input modalities and focusing on two primary output formats: procedural command sequences and boundary representation (B-rep) models [24, 10, 26]. While these methods advance the field, they often rely on task-specific command sequences that are not inherently supported by pre-existing language models, necessitating extensive de novo model training. Furthermore, the generated sequences typically require additional processing to yield actionable 3D models, adding undesirable complexity to the overall pipeline. We directly produce executable

Preprint. Under review.

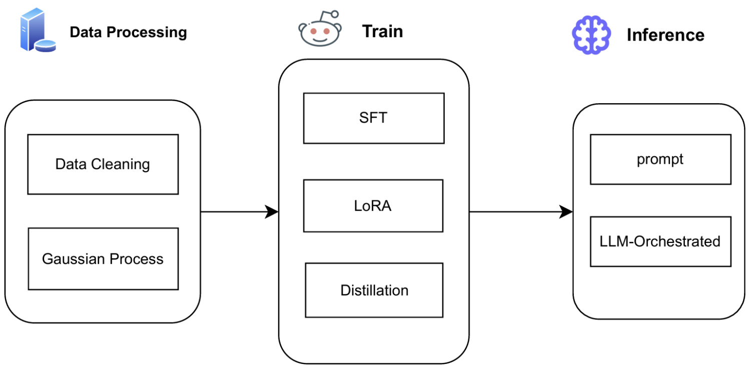

Figure 1: overview

code that can effortlessly render a 3D model. The code can be executed directly without any external software dependencies. We selected CadQuery [20] as our target representation. CadQuery further distinguishes itself with a clean, high-level Application Programming Interface (API) that encapsulates common CAD operations—like creating boxes, arcs, circles, and extrusions—into concise, human-readable code. This unique combination of executability and interpretability makes CadQuery an exceptionally suitable target for generative modeling with language models. Leveraging this representation offers two significant advantages. First, contemporary large language models already demonstrate remarkable proficiency in generating Python code; in many instances, they can directly synthesize CadQuery code. With their advanced capabilities in multimodal comprehension and spatial reasoning, sophisticated pretrained models are increasingly adept at translating high-level inputs directly into 3D-generative code. Second, the well-established model scaling laws suggest that larger pretrained models inherently exhibit superior performance [14]. By fine-tuning these robust models on appropriately paired input-CadQuery data, we can sidestep the prohibitive costs of training models from scratch while simultaneously achieving enhanced performance.

To thoroughly investigate this innovative approach, our work undertakes two critical phases. First, we build upon the Text2CAD dataset [15], which originally augments DeepCAD [21] with natural language descriptions of 3D models. We performed a thorough data cleaning and validation process, identifying and removing samples where the generated Python code was not executable or produced errors. This rigorous refinement results in a new, high-quality dataset comprising approximately 170,000 text–CadQuery pairs [23], enabling direct training for Text-to-CadQuery generation. Second, we undertake a comprehensive fine-tuning effort across pretrained model. Our chosen model is Qwen2.5-3B [27]. Our extensive experiments yield two pivotal insights. Furthermore, we consistently observe a clear scaling trend: as model size increases, so does generation accuracy. These compelling results strongly corroborate our two core assertions: (1) directly generating CadQuery not only simplifies the modeling pipeline but also achieves superior performance by capitalizing on the inherent capabilities of existing pretrained models in Python code generation and spatial reasoning; and (2) larger pretrained models unequivocally confer enhanced CAD generation abilities.

Building upon these significant findings, our main contributions are summarized as follows:

- We propose a data-efficient training strategy informed by **Gaussian Process Regression analysis**, which quantifies dataset redundancy. This allows for effective model training using only 30% of the original training data, enhancing efficiency and mitigating overfitting.
- We implement and evaluate advanced training techniques for Text-to-CAD models, including **Parameter-Efficient Fine-Tuning (PEFT) with LoRA and 4-bit quantization** for large models, and a **multi-sample knowledge distillation** approach where a student model learns from diverse outputs generated by a larger teacher model, thereby improving the student's performance and output variety.
- We develop an **LLM-Orchestrated hierarchical CAD generation pipeline** for complex, multi-part objects. This pipeline features: (a) LLM-driven assessment of complexity and decomposition of user prompts; (b) utilization of fine-tuned expert models for individual part generation; (c) LLM-based integration of these parts; and (d) a novel **iterative refinement**

loop where the orchestrating LLM revises generated CAD code based on execution error feedback, improving the robustness of the final output.

2 Related Work

Introduction: Text-to-X Generation: In recent years, Large Language Models (LLMs) have witnessed rapid advancements, with models like ChatGPT[7] and DeepSeek[4] showcasing remarkable capabilities across diverse domains. This progress has enabled the generation of various forms of content through natural language instructions, broadly termed as the Text-to-X paradigm, significantly expanding the application frontiers of artificial intelligence. Text-to-X generation aims to transform human natural language descriptions into concrete, structured outputs, such as Text-to-Image[8], Text-to-Code[19], and Text-to-3D[16].

Within this paradigm, Text-to-Image technologies, exemplified by models like DALL-E[18], have achieved notable success in synthesizing high-quality images from textual descriptions. Similarly, LLMs have demonstrated strong capabilities in code generation, with models such as Codex[1] accurately generating code from natural language prompts, a proficiency greatly enhanced by their pre-training on vast code corpora. However, the Text-to-X generation domain also faces common challenges, including ensuring the semantic accuracy and high fidelity of generated content, as well as maintaining creativity when confronted with complex or novel requirements.

Text-to-CAD (Text-to-Computer-Aided Design), as a crucial branch of the Text-to-X paradigm, is garnering increasing attention[29]. CAD is the industry standard for 3D modeling and plays a vital role in the design and development of products across various industries. As the complexity of modern designs increases, the potential for LLMs to enhance and streamline CAD workflows presents an exciting frontier. Traditionally, the design and creation of CAD models require specialized software and complex manual operations. The emergence of Text-to-CAD aims to directly generate CAD models from natural language descriptions, thereby significantly lowering design barriers and improving design efficiency[29].

Large Language Models (LLMs) in Computer-Aided Design: The integration of LLMs with Computer-Aided Design (CAD) is a rapidly evolving area. Despite the growing popularity of LLMs and the critical importance of CAD in modern industry, a comprehensive review focusing on their intersection was notably absent until recently. LLMs have demonstrated transformative potential in various generative and analytical tasks, including image synthesis[18], text generation, and code completion[1], making them promising candidates for advancing CAD through automation, intelligent design assistance, and semantic understanding of design specifications.

Current research in this area primarily leverages LLMs to generate intermediate representations rather than directly outputting 3D CAD models or 2D CAD drawings. These intermediate formats often include executable code, such as Python scripts, or parametric data structured as JSON files, which can then be parsed or executed to construct the final CAD models. Most state-of-the-art methods in this domain rely on multimodal inputs, particularly text and images, highlighting the increasing prevalence and potential of multimodal LLMs for CAD automation[29]. While other data formats like point clouds and sketch sequences are also being explored, the text and image modalities are the most frequently utilized. The GPT family of models, including GPT-4o, GPT-4, and GPT-4V, are the most frequently employed LLMs in CAD-related research, likely due to their superior performance[7].

Fine-tuning and Adaptation Techniques for LLMs: To adapt and optimize LLMs for specific downstream tasks like Text-to-CAD, fine-tuning and adaptation techniques play a crucial role. LLMs are first pre-trained on large textual datasets to acquire general language understanding and generation capabilities. This is then followed by fine-tuning, which involves further training the model on a smaller, task-specific dataset. Unlike pre-training which requires massive datasets, fine-tuning can be effective with tens or hundreds of thousands of high-quality samples.

Low-Rank Adaptation (LoRA) is a prominent parameter-efficient fine-tuning (PEFT) technique that has gained significant traction for adapting large pre-trained models to new tasks without retraining all their parameters[11]. For instance, in CAD-MLLM, Xu et al. [25] used LoRA during training to minimize learnable parameters when fine-tuning Vicuna-7B for parametric CAD generation. Similarly, Yuan et al. [28] leveraged the LoRA method to enable OpenECAD to generate CAD code effectively. Supervised Fine-tuning (SFT) is another common step in adapting LLMs, often using instruction-based data to refine the model's behavior. This process is critical for aligning LLMs with human intentions and ethical standards. BlenderLLM[5], is a script generation model that underwent SFT, fine-tuning the Qwen2.5-Coder-7B-Instruct model using a curated dataset and filtering high-quality generated data. The combination of SFT with techniques like LoRA allows for efficient and effective adaptation of powerful LLMs to specialized domains like CAD, enabling them to generate more accurate and contextually relevant outputs without the prohibitive costs of full model retraining. There are also other methods , such as DPO[17], RL[9], RLHF[30], and so on.

3 Method

3.1 Data Preprocessing

3.1.1 Data Cleaning

Our research utilizes the dataset constructed and annotated by Haoyang Xie [23] in the pioneering work. This dataset provides a robust foundation for our work. To further ensure its suitability and high quality for subsequent model training and evaluation, we applied an additional purification process. Specifically, we conducted compilation tests on every ground truth CAD query code within the original dataset and consequently removed samples whose CAD codes failed to compile. This critical procedure guarantees that our models are trained on, and evaluated against, syntactically correct and executable CAD programs.

3.1.2 Gaussian Process

An intuitive observation is the presence of redundancy within the dataset. Despite its considerable size (approximately 170,000 instances), we hypothesize that the Text-to-CadQuery dataset exhibits high internal similarity. At the output level, this is evidenced by the models' tendency to generate repetitive `scale` parameters (further examples are provided in Appendix A.2) and identical code comments. This strong similarity in outputs, likely extending to the embedding space, suggests a risk of severe overfitting, particularly for higher-capacity models. To empirically substantiate this hypothesis, we conducted a **Gaussian Process Regression** (GPR) analysis on `data_val.jsonl`, our validation set, which was randomly sampled from the training data and is thus assumed to share its distributional characteristics.

$$\hat{s}_{\mathcal{GP},v}(x_*) = \mathbf{k}(x_*, X_{D'})^T (K_{D'D'} + \sigma_{n,v}^2 I)^{-1} \mathbf{s}_{D',v}. \tag{1}$$

where the terms are defined as follows:

- $X_{D'} = \{x_j\}_{j=1}^M$ represents the set of M training input points.
- $\mathbf{k}(x_*, X_{D'})$ is a vector in $\mathbb{R}^M$ denoting the covariances between the new input x_* and each training input $x_j \in X_{D'}$, with its j-th element being $k(x_*, x_j)$.
- $K_{D'D'}$ is the $M \times M$ covariance matrix computed from the training inputs, where each entry $(K_{D'D'})_{ij} = k(x_i, x_j)$ is the kernel evaluation between $x_i, x_j \in X_{D'}$.
- $\sigma_{n,v}^2$ is the noise variance hyperparameter for the v-th logit dimension, accounting for potential observation noise or model misspecification.
- $\mathbf{s}_{D',v}$ is a vector in $\mathbb{R}^M$ containing the observed values of the v-th logit dimension from the training set D'.

We employed Gaussian Process analysis to quantitatively demonstrate the redundancy and high similarity among a large portion of the dataset. Based on this observation, we optimized the training procedure by randomly selecting only 30% of the training set for model training.

3.2 Training Phase

3.2.1 Parameter Efficient Fine-Tuning

We split the data into 90% for training, 5% for validation, and 5% for testing. Despite their varying parameter sizes, all models converge efficiently. For reference, Text2CAD Transformer trains a 23M decoder from scratch over two days on a single A100 GPU. In contrast, our smallest model (CodeGPT-small, 124M) fine-tunes in under an hour, and even our largest model (Mistral-7B) completes training within 33 hours (see Table 1). To adapt to different model sizes, we use full-parameter supervised fine-tuning (SFT) for smaller models, and parameter-efficient fine-tuning (PEFT) for larger models such as Mistral-7B. We implement PEFT using LoRA [12], described below.

LoRA for Large Models: LoRA inserts trainable low-rank matrices into the attention layers while keeping the base model weights frozen. Specifically, we use 4-bit quantization and configure LoRA with rank $r = 16$ and scaling factor $\alpha = 32$. The learned update is defined as:

$$\Delta W = AB, \quad A \in \mathbb{R}^{d\times r}, \quad B \in \mathbb{R}^{r\times d}, \quad r \ll d \tag{2}$$

where $W \in \mathbb{R}^{d\times d}$ is the original weight matrix and ΔW is the low-rank adaptation added during training. A and B are the learned low-rank matrices with rank r, where d is the hidden dimension of the model. This approach enables efficient adaptation of large models on limited hardware.

Table 1: Fine-tuning configurations for all models

Model	Training Hardware	Training Time	Epochs	Learning Rate	Batch Size	Techniques
Qwen2.5-3B	6×4090 24GB	15h 43m	3	5e-5	72	SFT
Qwen2.5-3B	6×4090 24GB	6h 15m	3	5e-5	72	LoRA+4bit

3.2.2 Model Distillation

To enhance the performance of a compact student model using knowledge distillation, we employ a teacher-student framework where the teacher is the larger Qwen3-32B model and the student is Qwen2.5-3B. The training data consists of 170,000 text-to-CAD instruction-output pairs. To mitigate overfitting and encourage diversity in the student's learning process, we adopt a multi-sample distillation strategy. Specifically, for each training example, we prompt the teacher model to generate 10 diverse responses using stochastic decoding (e.g., temperature sampling). During student training, for each input, we randomly sample one of these 10 teacher outputs as the supervision signal. This randomized response selection introduces natural variation during training and prevents the student from overfitting to deterministic outputs. This method is particularly suitable for tasks with multiple plausible correct outputs, such as text-to-CAD generation.

3.3 LLM-Orchestrated CAD

Our method generates CAD model code from textual descriptions by enhancing an existing text-to-CAD conversion framework that leverages fine-tuned expert models. We introduce an intelligent decomposition and robust integration pipeline, driven by a Large Language Model (LLM), specifically designed for multi-part complex objects. This approach boosts the system's capabilities and flexibility.

Baseline Approach: The baseline method we enhance relies on M_{expert}, a fine-tuned expert model. Mexpert directly processes a user-provided textual prompt, denoted as P_{user}, to generate corresponding CAD query code, Q_{CAD}.

$$Q_{CAD} = M_{\text{expert}}(P_{\text{user}}) \tag{3}$$

While this approach performs well for describing single, structurally simple objects, its expressive power and accuracy are limited when addressing complex, multi-part objects or scenarios that demand precise compositional logic.

LLM-Orchestrated CAD: To address the limitations of the baseline method, we propose a hierarchical CAD query generation pipeline 2. This pipeline centrally leverages the *Qwen2.5 7B model* for high-level semantic understanding, task decomposition, and code integration. Provided with the user's textual prompt P_{user} and a system prompt P_{system} – meticulously designed to guide its behavior – the Qwen2.5 7B model initially assesses whether the CAD model described in P_{user} is composed of multiple parts.

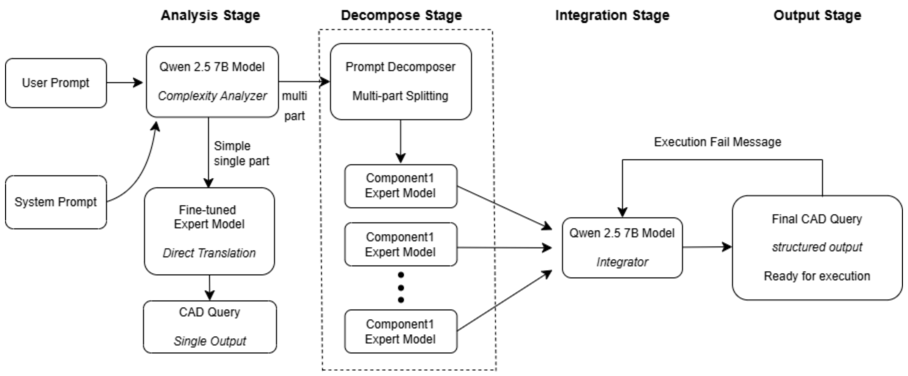

Figure 2: overview of LLM-Orchestrated CAD pipeline

$$\text{is\_multipart} = \text{Qwen2.5}(P_{\text{user}}, P_{\text{system\_assess}}) \tag{4}$$

Here, $P_{\text{system\_assess}}$ is the system prompt guiding the complexity assessment.

- If is_multipart evaluates to false (indicating a single-part model), the process reverts to the baseline method: the user prompt P_{user} is directly processed by the fine-tuned expert model M_{expert} to generate the final CAD query code Q_{final}.

$$Q_{\text{final}} = M_{\text{expert}}(P_{\text{user}}) \tag{5}$$

- If is_multipart evaluates to true (indicating a multi-part model), the subsequent multi-part processing pipeline is activated.

For CAD models identified as multi-part, the detailed process is as follows:

1. Guided by a dedicated system prompt, the Qwen2.5 7B model leverages its understanding of P_{user} to decompose the original user prompt into a set of sub-prompts, $\{p_1, p_2, \ldots, p_n\}$, each corresponding to a single logical part.
2. Each resulting sub-prompt p_j is individually processed by the fine-tuned expert model M_{expert} to generate the CAD query code q_j for the corresponding part.
3. The set of CAD query codes for individual parts, $\{p_1, p_2, \ldots, p_n\}$, is collected. These codes, along with a dedicated system prompt $P_{\text{system\_integrate}}$ for guiding integration, are provided to the Qwen2.5 7B model. The model's task is to synthesize these individual codes into a cohesive query code, $Q^{(0)}_{\text{final}}$, that describes the complete CAD model.

$$Q^{(0)}_{\text{final}} = \text{Qwen2.5}_{\text{integrate}}(\{q_1, q_2, \ldots, q_n\}, P_{\text{system\_integrate}}) \tag{6}$$

4. The generated code $Q^{(k)}_{\text{final}}$ (at iteration k) is executed in the target CAD software environment. If execution yields an error message $E^{(k)}$, this message, along with the current $Q^{(k)}_{\text{final}}$, is provided as feedback to the Qwen2.5 7B model. Guided by a dedicated system prompt $P_{\text{system\_refine}}$, the model then attempts to comprehend the error and revise $Q^{(k)}_{\text{final}}$, producing an updated CAD query code $Q^{(k+1)}_{\text{final}}$.

$$Q^{(k+1)}_{\text{final}} = \text{Qwen2.5}_{\text{refine}}(Q^{(k)}_{\text{final}}, E^{(k)}, P_{\text{system\_refine}}) \tag{7}$$

This iterative refinement loop continues until the CAD query code executes successfully or a predefined maximum number of attempts is reached. This closed-loop feedback mechanism enhances the robustness and usability of the final generated code.

4 Experiments and Evaluation

4.1 Evaluation Metrics

To compare our method fairly against the baseline Text2CAD Transformer, we adopt three evaluation metrics: Chamfer Distance (CD), Invalid Rate (IR), and Model Parameters. These metrics are used consistently across both methods to evaluate the quality of the generated 3D shapes.

Chamfer Distance (CD): Measures the geometric discrepancy between the generated and ground-truth 3D models [6]. For each STL mesh, we normalize its scale, uniformly sample 10,000 surface points, and compute the average squared distance between nearest neighbors across both point clouds:

$$\mathrm{CD}(P,Q) = \frac{1}{|P|}\sum_{p\in P}\min_{q\in Q}\|p-q\|^2 + \frac{1}{|Q|}\sum_{q\in Q}\min_{p\in P}\|q-p\|^2 \tag{8}$$

where P and Q are point sets sampled from the predicted and ground-truth shapes, respectively. Lower values indicate higher geometric fidelity.

Invalid Rate (IR): Captures the percentage of generated CadQuery scripts that fail during execution, due to syntax errors, incomplete expressions, or invalid operations. For example, an IR of 1.32 corresponds to 1.32% of the generated samples failing to produce a valid 3D shape. We provide a representative failure case and analysis in Appendix A.2 for reference.

Model Parameters: The models in our study exhibit distinct characteristics regarding their parameter counts, reflecting different architectural choices and optimization strategies. The Text2CAD Transformer is a bespoke architecture with a total of 363 million parameters. In contrast, both our Supervised Fine-Tuned (SFT) and Parameter-Efficient Fine-Tuned (PEFT) models are based on the Qwen2.5-3B large language model. This base model inherently contains 3 billion parameters.

Table 2: Comparison of models on Model Parameters, Chamfer Distance and Invalid Rate.

Model	**Model Parameters**	**Medium CD $\times 10^3\downarrow$**	**Mean CD $\times 10^3\downarrow$**	**IR$\downarrow$**
Text2CAD Transformer	363M	0.370	26.417	3.5
SFT Qwen2.5-3B	3B	0.192	10.238	**6.2**
PEFT Qwen2.5-3B	3B	0.219	15.943	6.5
Distill Qwen2.5-3B	3B	**0.185**	**10.182**	6.3
Qwen3-32B	32B	0.164	8.832	0.56

For our SFT Qwen2.5-3B model, the entire 3 billion parameters of the Qwen2.5 base model were fine-tuned. However, for our PEFT Qwen2.5-3B model, we employed **LoRA** (Low-Rank Adaptation) combined with **4-bit quantization**. This means that while the underlying Qwen2.5 base model still has 3 billion parameters, only a small, specific subset of these parameters was actively trained or adapted using LoRA. The 4-bit quantization further optimizes the model by reducing the precision of its weights, significantly lowering the memory footprint and accelerating inference, even though the total number of parameters remains consistent with the base model. This approach highlights the efficiency of PEFT methods in leveraging large pre-trained models with minimal training overhead and enhanced deployment capabilities.

4.2 Ablation Study

4.2.1 Model Architecture and Scale Evaluation

To thoroughly evaluate the efficacy of our approach, particularly regarding the Fast-CadQuery output format, we conducted an extensive ablation study across various model sizes and architectures.

Our findings reveal that even the smallest model, CodeGPT-small (124M), consistently surpasses the baseline performance across all metrics following fine-tuning. Interestingly, GPT-2 medium (355M), despite possessing nearly three times the parameters of CodeGPT-small, did not yield a significant improvement and, in some cases, exhibited a higher Invalid Rate. This performance disparity is likely attributable to CodeGPT's pre-training on Python code, which provides it with a substantial inductive bias for generating syntactically correct and semantically appropriate CadQuery scripts.

Furthermore, a critical observation from our study is the exceptional performance achieved by our quantized models, even when significantly reduced in size. This outcome strongly underscores that our method does not necessitate an exceptionally large or powerful pre-trained model to achieve high performance. Instead, it demonstrates the remarkable efficiency and effectiveness of our approach, proving its capability to deliver robust CAD generation even under resource-constrained conditions.

Across models of increasing capacity, we generally observe a clear upward trend in performance, indicating that greater model size correlates with an enhanced ability to generate more accurate and executable CadQuery programs. Our fine-tuned Qwen2.5-3B model achieves the best overall results, with a Chamfer Distance of 0.192 (median) and 12.513 (mean) after $\times 10^3$. These results collectively demonstrate state-of-the-art performance for Text-to-CAD generation using symbolic CadQuery output. Detailed performance metrics are provided in B, shown in Table 3.

4.2.2 Dataset Analysis and Redundancy

As noted in Section 4.2.1, a counterintuitive observation emerged: the Mistral-7B model, despite its larger parameter count, underperformed the Qwen2.5-3B model. Specifically, its Chamfer Distance (CD) scores (mean CD lower than SFT Qwen2.5-3B, and median CD ranking second to last among evaluated models) were more comparable to those of the significantly smaller Gemma-1.1-2B model. We attribute this unexpected outcome primarily to the **suboptimal quality and high redundancy within the training data**, which likely curtails the larger model's ability to effectively learn the nuanced, task-specific generation patterns essential for precise 3D object modeling.

We trained GPR models using varying proportions of this validation set to model the underlying distribution of output embeddings. As depicted in Figure 3, a GPR model trained on **merely ~5.44%** of the validation data (300 out of 5,513 samples) was capable of generating an output embedding distribution that closely approximated the empirical distribution derived from the entire validation set.

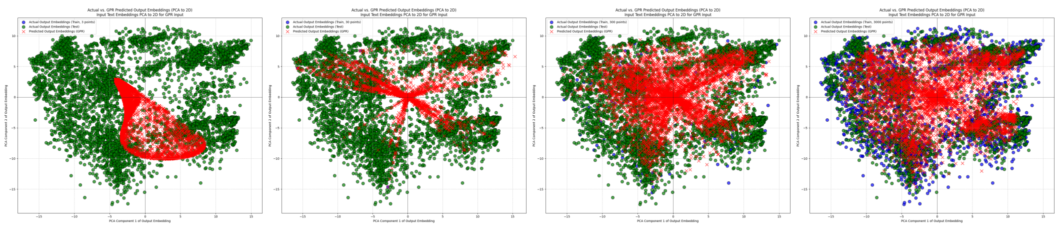

Figure 3: GPR analysis of output embedding distributions on the validation set (total 5,513 samples). The figure illustrates how the GPR-learned distribution increasingly approximates the true empirical distribution of output embeddings as the number of training samples increases. From left to right, GPR models were trained using 3, 30, 300, and 3,000 randomly selected samples, respectively. Each panel shows the 2D PCA projection of the GPR-generated output embeddings (red 'x') overlaid with the true output embeddings from the validation set (blue/green points).

This result strongly indicates significant redundancy within the Text-to-CadQuery dataset. It supports our conjecture that the larger Mistral-7B model's performance was hampered by this data characteristic, making it prone to overfitting these prevalent, yet potentially superficial, patterns. Consequently, this analysis lends credence to our strategy of employing LoRA with 4-bit quantization. The inherent regularization effects of these techniques—possibly by introducing beneficial noise or by constraining effective model capacity—may help mitigate overfitting and enhance generalization when training on datasets with such pronounced redundancy.

4.2.3 Detailed Geometric Performance Analysis

In addition to the three core metrics used for comparison, we introduce two supplementary metrics, F1 Score and Volumetric IoU, to further evaluate the geometric fidelity of generated shapes. Both metrics offer complementary perspectives: F1 captures local geometric alignment, while IoU assesses global volumetric overlap. Below, we describe each in detail.

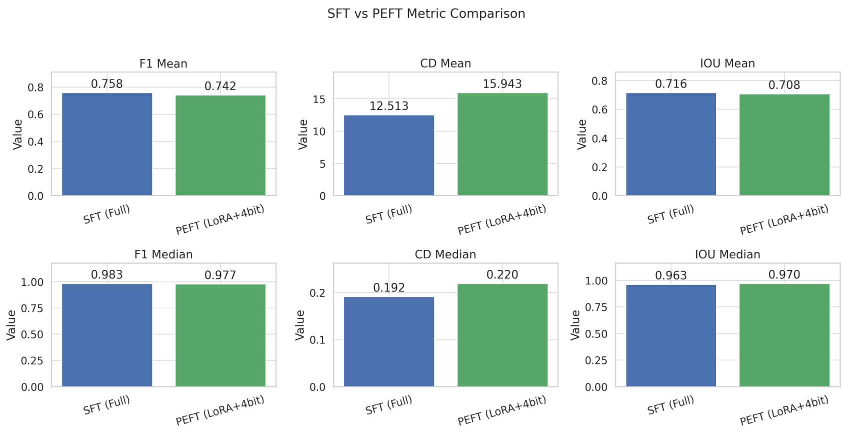

Figure 4: Performance comparison between full-precision SFT and our PEFT method (LoRA + 4-bit). Each subfigure reports the mean and median values for three key evaluation metrics: F1 score and Intersection over Union (higher is better), Chamfer Distance (lower is better). The results demonstrate that PEFT achieves competitive performance with significantly reduced computational cost.

F1 Score: We adopt a point-based F1 score to evaluate the alignment between predicted and ground-truth meshes. A predicted point is considered correct if its nearest neighbor in the ground-truth set lies within the threshold, and vice versa. The F1 score is then calculated as:

$$\mathrm{F1} = \frac{2 \cdot \mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{9}$$

Volumetric IoU: To further assess 3D shape similarity, we compute the volumetric Intersection over Union (IoU) between voxelized versions of the predicted and ground-truth meshes [22]. Let V_1 and V_2 denote the occupied voxel grids. IoU is computed as:

$$\mathrm{IoU} = \frac{|V_1 \cap V_2|}{|V_1 \cup V_2|} \tag{10}$$

This metric captures global shape overlap and is particularly useful for assessing coarse structural fidelity. Padding is applied to align voxel grid sizes when necessary.

As illustrated in Figure 4, our proposed PEFT approach, which integrates LoRA with 4-bit quantization, demonstrates performance that is highly comparable to the full-precision SFT baseline across key evaluation metrics, including F1 score, Chamfer Distance, and IoU. Remarkably, this efficiency is achieved while providing a $3\times$ speedup in training and inference, highlighting the effectiveness of low-rank adaptation and quantized representation in reducing computational overhead without significantly compromising performance.

5 Conclusion

In this work, we present a novel approach to symbolic 3D model generation by leveraging the high-level CadQuery API, enabling direct translation from natural language to executable 3D CAD programs. By extensively cleaning and curating the existing Text2CAD dataset, we constructed a high-quality corpus of approximately 170,000 samples suitable for robust training. Our experiments demonstrate that scaling up language models significantly improves generation quality, with the Qwen2.5-3B model yielding strong performance. Additionally, we explore and validate several effective training strategies, including supervised fine-tuning, LoRA-based parameter-efficient adaptation, and model distillation, all of which contribute to enhanced model capability. Our findings reveal that direct CadQuery code generation not only simplifies the overall CAD pipeline but also aligns well with pretrained models' strengths in code synthesis and spatial reasoning. This work lays the foundation for more accessible and scalable CAD tools, reducing the entry barrier for non-expert users and opening new opportunities for automation in engineering, design, and digital fabrication.

References

[1] Giulio Aielli, Juliette Alimena, Saul Balcarcel-Salazar, Eli Ben Haim, András Barnabás Burucs, Roberto Cardarelli, Matthew J. Charles, Xabier Cid Vidal, Albert De Roeck, Biplab Dey, Silviu Dobrescu, Ozgur Durmus, Mohamed Elashri, Vladimir V. Gligorov, Rebeca Gonzalez Suarez, Zarria Gray, Conor Henderson, Louis Henry, Philip Ilten, Daniel Johnson, Jacob Kautz, Simon Knapen, Bingxuan Liu, Yang Liu, Saul López Soliño, Pablo Eduardo Menéndez-Valdés Pérez, Titus Mombächer, Benjamin Nachman, David T. Northacker, Gabriel M. Nowak, Michele Papucci, Gabriella Pásztor, María Pereira Martínez, Michael Peters, Jake Pfaller, Luca Pizzimento, Máximo Pló Casasús, Gian Andrea Rassati, Dean J. Robinson, Emilio Xosé Rodríguez Fernández, Debashis Sahoo, Sinem Simsek, Michael D. Sokoloff, Joeal Subash, James Swanson, Abhinaba Upadhyay, Riccardo Vari, Carlos Vázquez Sierra, Gábor Veres, Nigel Watson, Michael K. Wilkinson, Michael Williams, and Eleanor Winkler. Codex-b: Opening new windows to the long-lived particle frontier at the lhc, 2025. URL `https://arxiv.org/abs/2505.05952`.

[2] Autodesk. Autocad 2025 release notes. `https://help.autodesk.com/cloudhelp/2025/ENU/AutoCAD-ReleaseNotes/files/AUTOCAD_2025_RELEASE_NOTES.html`, 2024. Accessed: 2025-05-01.

[3] Johannes Brozovsky, Nathalie Labonnote, and Olli Vigren. Digital technologies in architecture, engineering, and construction. *Automation in Construction*, 158:105212, 2024.

[4] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

[5] Yuhao Du, Shunian Chen, Wenbo Zan, Peizhao Li, Mingxuan Wang, Dingjie Song, Bo Li, Yan Hu, and Benyou Wang. Blenderllm: Training large language models for computer-aided design with self-improvement, 2024. URL `https://arxiv.org/abs/2412.14203`.

[6] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[7] Tao Fang, Shu Yang, Kaixin Lan, Derek F Wong, Jinpeng Hu, Lidia S Chao, and Yue Zhang. Is chatgpt a highly fluent grammatical error correction system? a comprehensive evaluation. *arXiv preprint arXiv:2304.01746*, 2023.

[8] Uri Gadot, Rinon Gal, Yftah Ziser, Gal Chechik, and Shie Mannor. Policy optimized text-to-image pipeline design, 2025. URL `https://arxiv.org/abs/2505.21478`.

[9] Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning, 2024. URL `https://arxiv.org/abs/2403.04642`.

[10] Negar Heidari and Alexandros Iosifidis. Geometric deep learning for computer-aided design: A survey. *arXiv preprint arXiv:2402.17695*, 2024.

[11] 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, 2021. URL `https://arxiv.org/abs/2106.09685`.

[12] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

[13] Bonsa Regassa Hunde and Abraham Debebe Woldeyohannes. Future prospects of computer-aided design (cad)–a review from the perspective of artificial intelligence (ai), extended reality, and 3d printing. *Results in Engineering*, 14:100478, 2022.

[14] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[15] Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad models from beginner-to-expert level text prompts. *arXiv preprint arXiv:2409.17106*, 2024.

[16] Lu Ling, Chen-Hsuan Lin, Tsung-Yi Lin, Yifan Ding, Yu Zeng, Yichen Sheng, Yunhao Ge, Ming-Yu Liu, Aniket Bera, and Zhaoshuo Li. Scenethesis: A language and vision agentic framework for 3d scene generation, 2025. URL `https://arxiv.org/abs/2505.02836`.

[17] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL `https://arxiv.org/abs/2305.18290`.

[18] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021. URL `https://arxiv.org/abs/2102.12092`.

[19] Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. Execution-based code generation using deep reinforcement learning, 2023. URL `https://arxiv.org/abs/2301.13816`.

[20] Jeremy Wright, thebluedirt, Marcus Boyd, Lorenz, Innovations Technology Solutions, Hasan Yavuz ÖZDERYA, Bruno Agostini, Jojain, Michael Greminger, Seth Fischer, Justin Buchanan, cactrot, huskier, Ruben, Iulian Onofrei, Miguel Sánchez de León Peque, Martin Budden, Hecatron, Peter Boin, Wink Saville, Pavel M. Penev, Bryan Weissinger, M. Greyson Christoforo, Jack Case, AGD, Paul Jurczak, nopria, moeb, and jdegenstein. Cadquery/cadquery: Cadquery 2.4.0, January 2024. URL `https://doi.org/10.5281/zenodo.10513848`.

[21] 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.

[22] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[23] Haoyang Xie and Feng Ju. Text-to-cadquery: A new paradigm for cad generation with scalable large model capabilities, 2025. URL `https://arxiv.org/abs/2505.06507`.

[24] Jingwei Xu, Zibo Zhao, Chenyu Wang, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm. *arXiv preprint arXiv:2411.04954*, 2024.

[25] Jingwei Xu, Zibo Zhao, Chenyu Wang, Wen Liu, Yi Ma, and Shenghua Gao. Cad-mllm: Unifying multimodality-conditioned cad generation with mllm, 2025. URL `https://arxiv.org/abs/2411.04954`.

[26] Qun-Ce Xu, Tai-Jiang Mu, and Yong-Liang Yang. A survey of deep learning-based 3d shape generation. *Computational Visual Media*, 9(3):407–442, 2023.

[27] 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.

[28] Zhe Yuan, Jianqi Shi, and Yanhong Huang. Openecad: An efficient visual language model for editable 3d-cad design. *Computers amp; Graphics*, 124:104048, November 2024. ISSN 0097-8493. doi: 10.1016/j.cag.2024.104048. URL `http://dx.doi.org/10.1016/j.cag.2024.104048`.

[29] Licheng Zhang, Bach Le, Naveed Akhtar, Siew-Kei Lam, and Tuan Ngo. Large language models for computer-aided design: A survey, 2025. URL `https://arxiv.org/abs/2505.08137`.

[30] Yinmin Zhong, Zili Zhang, Bingyang Wu, Shengyu Liu, Yukun Chen, Changyi Wan, Hanpeng Hu, Lei Xia, Ranchen Ming, Yibo Zhu, and Xin Jin. Optimizing rlhf training for large language models with stage fusion, 2025. URL `https://arxiv.org/abs/2409.13221`.

A Technical Appendices and Supplementary Material

A.1 Prompt Design

```
Write CAD sequence json to CadQuery.  Here are two example:
```

1. CAD Sequence (JSON)

```
{
  "final_name": "Cylinder",
  "parts": {
    "part_1": {
      "coordinate_system": {
        "Euler Angles": [0.0, 0.0, 0.0],
        "Translation Vector": [0.0, 0.0, 0.0]
      },
      "sketch": {
        "face_1": {
          "loop_1": {
            "circle_1": {
              "Center": [0.375, 0.375],
              "Radius": 0.375
            }
          }
        }
      },
      "extrusion": {
        "extrude_depth_towards_normal": 0.1046,
        "extrude_depth_opposite_normal": 0.0,
        "sketch_scale": 0.75,
        "operation": "NewBodyFeatureOperation"
      },
      "description": {
        "name": "Cylinder",
        "length": 0.7499999633140781,
        "width": 0.7499999633140781,
        "height": 0.10461455694430137
      }
    }
  }
}
```

1. CadQuery Code

```python
import cadquery as cq
from cadquery.vis import show

# --- Part 1: Cylinder ---
part_1_radius = 0.375 * 0.75  # Sketch radius scaled
part_1_height = 0.1046

part_1 = cq.Workplane("XY") \
    .circle(part_1_radius) \
    .extrude(part_1_height)

# --- Final Result ---
result = part_1
cq.exporters.export(result, 'result.stl')
```

2. CAD Sequence (JSON)

```
{
  "final_name": "Rectangular prism with a curved top and a cutout on one side",
  "parts": {
    "part_1": {
      "coordinate_system": {
        "Euler Angles": [0.0, 0.0, -90.0],
        "Translation Vector": [0.0, 0.5625, 0.0]
      },
      "sketch": {
        "face_1": {
          "loop_1": {
            "line_1": {
              "Start Point": [0.0, 0.0],
              "End Point": [0.75, 0.0]
            },
            "line_2": {
              "Start Point": [0.75, 0.0],
              "End Point": [0.75, 0.0625]
            },
            "line_3": {
              "Start Point": [0.75, 0.0625],
              "End Point": [0.5625, 0.0625]
            },
            "line_4": {
              "Start Point": [0.5625, 0.0625],
              "End Point": [0.5625, 0.4531]
            },
            "line_5": {
              "Start Point": [0.5625, 0.4531],
              "End Point": [0.5313, 0.4531]
            },
            "arc_1": {
              "Start Point": [0.5313, 0.4531],
              "Mid Point": [0.375, 0.2969],
              "End Point": [0.2188, 0.4531]
            },
            "line_6": {
              "Start Point": [0.2188, 0.4531],
              "End Point": [0.1875, 0.4531]
            },
            "line_7": {
              "Start Point": [0.1875, 0.4531],
              "End Point": [0.1875, 0.0625]
            },
            "line_8": {
              "Start Point": [0.1875, 0.0625],
              "End Point": [0.0, 0.0625]
            },
            "line_9": {
              "Start Point": [0.0, 0.0625],
              "End Point": [0.0, 0.0]
            }
          }
        }
      },
      "extrusion": {
        "extrude_depth_towards_normal": 0.5625,
        "extrude_depth_opposite_normal": 0.0,
        "sketch_scale": 0.75,
        "operation": "NewBodyFeatureOperation"
      },
      "description": {
        "name": "Rectangular prism with a curved top and a cutout on one side",
        "length": 0.7500000000000001,
        "width": 0.45312500000000006,
        "height": 0.5625000000000001
      }
    }
  }
}
```

2. CadQuery Code

```python
import cadquery as cq
from cadquery.vis import show

# --- Part 1 ---
part_1 = (
    cq.Workplane("XY")
    .moveTo(0.0, 0.0)
    .lineTo(0.75, 0.0)
    .lineTo(0.75, 0.0625)
    .lineTo(0.5625, 0.0625)
    .lineTo(0.5625, 0.4531)
    .lineTo(0.5313, 0.4531)
    .threePointArc((0.375, 0.2969), (0.2188, 0.4531))
    .lineTo(0.1875, 0.4531)
    .lineTo(0.1875, 0.0625)
    .lineTo(0.0, 0.0625)
    .lineTo(0.0, 0.0)
    .close()
    .extrude(0.5625)
)

# --- Coordinate System Transformation for Part 1 ---
part_1 = part_1.rotate((0, 0, 0), (0, 0, 1), -90)
part_1 = part_1.translate((0, 0.5625, 0))

# --- Assembly ---
assembly = part_1
cq.exporters.export(assembly, "output.stl")
```

```
Your output show only contain executable python code, start with import.
```

A.2 Failure Case Analysis

Despite the overall robust performance of our model, we identified a few categories of failure cases during our experiments. These primarily stem from the inherent complexity of generating precise CAD models from textual descriptions and the limitations of sequence-to-sequence modeling.

One notable failure mode arises from **code truncation due to sequence length limitations**. For particularly complex CAD designs, the corresponding `cadquery` code can become exceedingly long. To manage computational resources and maintain practical inference times, we imposed a maximum sequence length on the generated code. After evaluating the trade-off between execution efficiency, computational cost, and the number of resulting unexecutable code instances, we selected a `max_length` of 1024 tokens. While this threshold accommodates the vast majority of cases, it occasionally leads to the truncation of code for very intricate designs, rendering them incomplete or unexecutable.

Another category of failure involves **anomalous scaling factors in the generated code**. In these instances, the model produces syntactically correct `cadquery` code that executes without explicit rendering errors. However, the resulting 3D model deviates significantly from the intended geometry due to inappropriate scaling applied to various dimensions. This issue appears to be a consequence of the model partially memorizing or overfitting to specific scale parameters present in the training dataset. While the rendered object might be topologically similar to the target, its proportions and absolute dimensions are incorrect. We have been actively working on mitigating this by incorporating more diverse data augmentation techniques, specifically focusing on randomizing scale and aspect ratios during training to improve the model's generalization capabilities regarding object dimensions.

An example of code exhibiting such scaling issues is provided below. Note the multiplication of base dimensions by seemingly arbitrary factors (e.g., '0.1149 * 0.1149' for 'part_1_length'). While the code runs, the scaling factors, if not aligned with the intended design logic or a consistent unit system, lead to a distorted output.

Failure code example with anomalous scaling

Listing 1: Anomalous Scaling in Generated Code

```
import cadquery as cq
import math
# from cadquery.vis import show # Assuming show is for local debugging

# --- Part 1: Rectangular Prism ---
part_1_length = 0.1149 * 0.1149  # Scaled length
part_1_width = 0.0718 * 0.1149   # Scaled width
part_1_height = 0.0431

part_1 = (
    cq.Workplane("XY")
    .rect(part_1_length, part_1_width)
    .extrude(part_1_height)
)

# --- Coordinate System Transformation for Part 1 ---
part_1 = part_1.rotate((0, 0, 0), (0, 0, 1), -90)
part_1 = part_1.translate((0, 0.0431, 0))

# --- Part 2: Cut Feature ---
part_2_length = 0.2011 * 0.2011  # Scaled length
part_2_width = 0.0172 * 0.2011   # Scaled width
part_2_height = 0.0057

part_2 = (
    cq.Workplane("XY")
    .moveTo(0, 0)
    .lineTo(part_2_length, 0)
    .threePointArc((part_2_length/2, 0.0172 * 0.2011), (0, 0))
    .close()
    .extrude(-part_2_height)
)

# --- Coordinate System Transformation for Part 2 ---
part_2 = part_2.rotate((0, 0, 0), (0, 0, 1), -90)
part_2 = part_2.rotate((0, 0, 0), (1, 0, 0), 180)
part_2 = part_2.translate((0.1575, 0.0431, 0.0278))

# --- Part 3: Base Cylinder ---
part_3_length = 0.4841 * 0.4841  # Scaled length
part_3_width = 0.1152 * 0.4841   # Scaled width
part_3_height = 0.0345

part_3 = (
    cq.Workplane("XY")
    .moveTo(0, 0)
    .lineTo(part_3_length, 0)
    .threePointArc((part_3_length/2, 0.1152 * 0.4841), (0, 0))
    .close()
    .extrude(part_3_height)
)

# --- Coordinate System Transformation for Part 3 ---
part_3 = part_3.rotate((0, 0, 0), (0, 0, 1), -90)
part_3 = part_3.translate((0.2659, 0.0431, 0))

# --- Assembly ---
assembly = part_1.cut(part_2).union(part_3)

# --- Export to STL ---
# Path was truncated !
cq.exporters.export(assembly, './
# Path was truncated !
cq.exporters.export(assembly, './stlcq/0000/00000007.stl')
```

Further research is focused on improving the model's understanding of geometric primitives and their compositional semantics, as well as developing more robust decoding strategies to handle length constraints and prevent dimensional inconsistencies.

B Supplementary of Ablation Study

Table 3: Comparison of models on Model Parameters, Chamfer Distance and Invalid Rate.

Model	**Model Parameters**	**Medium CD $\times 10^3\downarrow$**	**Mean CD $\times 10^3\downarrow$**	**IR$\downarrow$**
SFT CodeGPT small	124M	0.244	13.520	9.5
SFT GPT-2 medium	355M	0.221	12.575	15.6
SFT GPT-2 large	774M	0.209	12.175	13.9
SFT Gemma-3-1B	1B	0.207	12.012	7.7
LoRA Mistral-7B	7B	0.223	11.465	1.32
SFT Qwen2.5-3B	3B	0.192	10.238	6.2
PEFT Qwen2.5-3B	3B	0.219	15.943	6.5
Distill Qwen2.5-3B	3B	0.185	10.182	6.3
Qwen3-32B	32B	**0.164**	**8.832**	**0.56**