

CAD-Editor: A Locate-then-Infill Framework with Automated Training Data Synthesis for Text-Based CAD Editing

Yu Yuan^{1†} Shizhao Sun² Qi Liu¹ Jiang Bian²

Abstract

Computer Aided Design (CAD) is indispensable across various industries. *Text-based CAD editing*, which automates the modification of CAD models based on textual instructions, holds great potential but remains underexplored. Existing methods primarily focus on design variation generation or text-based CAD generation, either lacking support for text-based control or neglecting existing CAD models as constraints. We introduce *CAD-Editor*, the first framework for text-based CAD editing. To address the challenge of demanding triplet data with accurate correspondence for training, we propose an automated data synthesis pipeline. This pipeline utilizes design variation models to generate pairs of original and edited CAD models and employs Large Vision-Language Models (LVLMs) to summarize their differences into editing instructions. To tackle the composite nature of text-based CAD editing, we propose a locate-then-infill framework that decomposes the task into two focused sub-tasks: locating regions requiring modification and infilling these regions with appropriate edits. Large Language Models (LLMs) serve as the backbone for both sub-tasks, leveraging their capabilities in natural language understanding and CAD knowledge. Experiments show that *CAD-Editor* achieves superior performance both quantitatively and qualitatively. The code is available at <https://github.com/microsoft/CAD-Editor>.

[†] Work done during the internship at Microsoft Research Asia. Open-source research project starts from March 2024. Yu Yuan: <yyhappier@mail.ustc.edu.cn>, Qi Liu: <qiliuql@ustc.edu.cn>, Jiang Bian: <jiabia@microsoft.com>. ¹University of Science and Technology of China ²Microsoft Research Asia. Correspondence to: Shizhao Sun <shizsu@microsoft.com>.

Proceedings of the 42nd International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

1. Introduction

In the modern digital era, Computer-Aided Design (CAD) has become indispensable across industries. Most modern CAD tools follow the **Sketch-and-Extrude (SE) Operations** paradigm (Shahin, 2008; Camba et al., 2016), where designers sketch 2D curves to define the outer and inner boundaries of profiles, extrude them into 3D shapes, and combine these shapes to create complex models.

CAD model creation is an iterative process, where an initial draft undergoes multiple modifications until it aligns with user requirements. Natural language plays a crucial role throughout this process, serving as a key medium of communication. For non-experts, it offers the most intuitive way to express their needs, while for professionals, it enables fast, detailed, and precise instructions. Consequently, a system capable of automatically editing CAD models based on textual instructions – known as **text-based CAD editing** (Figure 1) – has the potential to revolutionize the entire CAD design workflow. Such a system could significantly accelerate the development of CAD models and empower a broader range of individuals, especially those with limited design expertise, to create CAD models more effectively.

While important, text-based CAD editing receives limited attention. Some studies explore *design variation generation*, where new CAD models are generated by randomly altering components of an existing model (Wu et al., 2021; Xu et al., 2022; 2023; Zhang et al., 2024b). However, these approaches lack support for text-based control over the appearance of the generated CAD models, limiting their practical usability. Another line of research makes initial attempts at *text-based CAD generation*, focusing on generating new CAD models directly from textual descriptions (Khan et al., 2024b; Li et al., 2024). Nonetheless, these methods do not incorporate an existing CAD model as input, which prevents them from leveraging the original design’s context and constraints.

Text-based CAD editing presents several distinct challenges. First, training for this task requires *triplet data with accurate correspondence* among an original CAD model, an editing instruction, and an edited CAD model. However, such data does not naturally exist, and manually collection

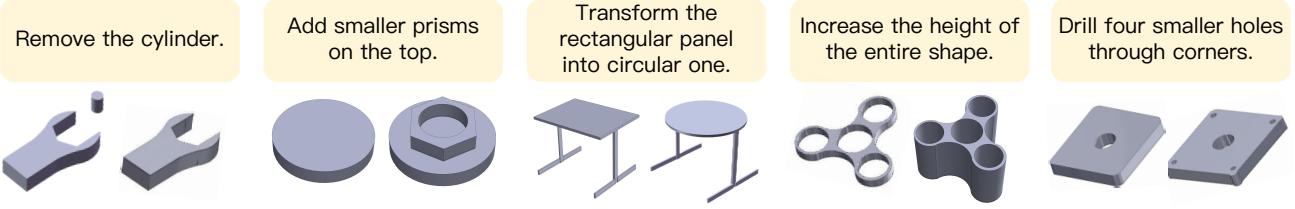


Figure 1. Text-based CAD editing achieved by CAD-Editor. Each sub-figure shows the editing instruction at the top, the original CAD model on the left, and the edited CAD model on the right. The rendered image is shown for better comprehension. The actual editing occurs on sketch-and-extrusion (SE) operations of a CAD model to provide editability and reusability.

is both costly and difficult to scale. Second, text-based CAD editing is inherently a *composite problem*. It demands a comprehensive understanding of diverse textual instructions and geometric concepts, the ability to locate the corresponding parts within the intricate structure of the CAD’s SE operations, and the capability to generate concrete modifications to these SE operations.

In this work, we introduce **CAD-Editor**, the first framework for text-based CAD editing. We frame the task as a sequence-to-sequence (seq2seq) generation problem, where the input combines an editing instruction and the sequence representation of the original CAD model, and the output is the sequence of the edited CAD model (Figure 2). To address the need for triplet data with accurate correspondence, we propose an *automated data synthesis pipeline* that leverages existing design variation models and Large Vision-Language Models (LVLMs) (Figure 3). Starting from an existing CAD model, design variation models generate edited CAD models by randomly altering parts while keeping others unchanged, producing pairs of original and edited CAD models. LVLMs then summarize the differences between these CAD models into editing instructions, resulting in triplets with strong correspondence. To tackle the composite nature of text-based CAD editing, we decompose the task into *specialized sub-tasks*: *locating* and *infilling*, each addressing a specific aspect of the editing process (Figure 4). For both sub-tasks, Large Language Models (LLMs) serve as the backbone, leveraging their strong natural language understanding capabilities and basic CAD-related knowledge, including SE operations and geometric concepts (Makatura et al., 2023). In the locating stage, LLMs identify regions requiring modification by generating a masked CAD sequence, where special tokens `<mask>` indicate the regions to be modified. In the infilling stage, LLMs generate appropriate edits for these masked regions, using the masked CAD sequence from the locating stage as context.

The contributions of this work are summarized as follows:

- We introduce a new task, text-based CAD editing, enabling precise edits through textual instructions to bet-

ter align with real-world user needs.

- We propose an automated data synthesis pipeline that combines design variation models and LVLMs to generate triplet data with accurate correspondence, addressing a critical challenge in training.
- We develop a locate-then-infill framework that decomposes the task into focused sub-tasks, and leverage LLMs as the backbone, effectively handling the composite nature of text-based CAD editing.
- We conduct extensive experiments, demonstrating that our approach outperforms baselines in generation validity, text-CAD alignment, and overall quality.

2. Related Works

CAD Generation. Parametric CAD (Shahin, 2008; Camba et al., 2016), defined by its sketch-and-extrude operations, is central to mechanical design due to its ability to retain modeling history, which facilitates both editing and manufacturing. Recent large-scale CAD datasets (Wu et al., 2021) have fueled the development of generative models. Wu et al. (2021) explored unconditional generation, where a random latent vector is used as input to generate CAD models. Xu et al. (2022; 2023); Zhang et al. (2024b) focused on design variation generation, which randomly modifies specific part of an existing CAD model. Recently, Khan et al. (2024a); Li et al. (2024) studied text-based CAD generation, which transforms textual descriptions into CAD models. Our work differs in two key aspects. First, we target a distinct task, unlike prior work, which either lacks text-based control (Wu et al., 2021; Xu et al., 2022; 2023; Zhang et al., 2024b) or disregards existing CAD models as constraints (Khan et al., 2024a; Li et al., 2024). Second, we introduce a novel locate-then-infill framework based on LLMs to handle the composite nature of text-based CAD editing. Previous approaches either rely on VAE-based (Wu et al., 2021; Xu et al., 2022; 2023) or transformer-based (Khan et al., 2024a; Li et al., 2024) architectures, or apply LLMs without accounting for task-specific needs (Zhang et al., 2024b).

Large Language Models (LLMs). Recently, LLMs like GPT (Achiam et al., 2023; OpenAI, 2023) and LLaMA (Tou-

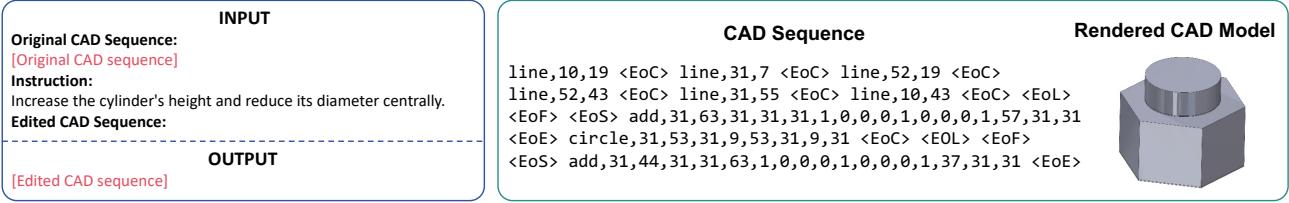


Figure 2. **Left:** Example input and output for CAD-Editor. The input combines the original CAD sequence with the editing instruction, and the output is the edited CAD sequence. The specific CAD sequence is shortened to ‘[Original (or Edited) CAD Sequence]’ to save space. **Right:** An illustration for a specific CAD sequence and its rendered CAD model.

vron et al., 2023) have distinguished themselves from smaller models, particularly through advanced prompting (Brown et al., 2020; Wei et al., 2022b; Kojima et al., 2022) or fine-tuning methods (Wei et al., 2022a). Beyond excelling in natural language processing (Yue et al., 2024; Zhan et al., 2025; Zhao et al., 2025), LLMs have transform generative tasks in other domains, e.g. motion generation (Zhang et al., 2024a) and material generation (Gruver et al., 2024). These advancements inspire us to adopt LLMs as the backbone for subtasks in our locate-then-infill framework. Moreover, LLMs and LVLMs are increasingly utilized for data synthesis to enhance training (Xu et al., 2024; Yu et al., 2024). Specifically, Khan et al. (2024b) leverage LLMs/LVLMs to synthesize data for text-based CAD generation. However, our task of text-based CAD editing presents distinct challenges. First, unlike Khan et al. (2024a), who generates two-tuple data (a text prompt and a CAD model), our task involves creating triplet data: an editing instruction, an original CAD model, and an edited CAD model. We address this by combining design variation models with LLMs/LVLMs. Second, while Khan et al. (2024a) focus on captioning single CAD models, our task requires summarizing differences between two CAD models. We handle this by introducing a stepwise captioning strategy.

Text-based Editing in Other Domains. Text-based editing has been widely explored across various domains, e.g., 3D editing (Mikaeili et al., 2023), image editing (Meng et al., 2021; Brooks et al., 2023), and video editing (Chai et al., 2023; Ceylan et al., 2023). It enables users to specify and modify particular objects or attributes with precision and flexibility. Inspired by these advancements, we introduce text-based editing in the CAD domain for the first time.

3. Approach Overview

Let \mathcal{I} denotes the editing instruction, $\mathcal{C}_{\text{orig}}$ the original CAD model and $\mathcal{C}_{\text{edit}}$ the edited CAD model. Here, the CAD model is represented using Sketch-and-Extrude (SE) operations, as this representation preserves the modeling history, making it easier to edit. The goal of text-based CAD editing is to learn a function f that takes \mathcal{I} and $\mathcal{C}_{\text{orig}}$ as inputs and

generates $\mathcal{C}_{\text{edit}}$ as output, i.e., $\mathcal{C}_{\text{edit}} = f(\mathcal{I}, \mathcal{C}_{\text{orig}})$.

We formulate text-based CAD editing as a sequence-to-sequence (seq2seq) generation problem. To achieve this, both the editing instruction and the CAD models are represented as sequences of textual tokens. The editing instruction \mathcal{I} naturally consists of textual tokens. For the CAD models $\mathcal{C}_{\text{orig}}$ and $\mathcal{C}_{\text{edit}}$, we adopt the sequence format introduced by Zhang et al. (2024b), which abstracts all primitives in SE operations, including numerical and categorical parameters, into textual tokens (Figure 2).

To address the need for training data with good correspondence among \mathcal{I} , $\mathcal{C}_{\text{orig}}$, and $\mathcal{C}_{\text{edit}}$, denoted as $\mathbb{D} = \{(\mathcal{I}, \mathcal{C}_{\text{orig}}, \mathcal{C}_{\text{edit}})\}_1^N$ (where N is the data size), we introduce an automated data synthesis pipeline (Section 4). First, we obtain $\mathcal{C}_{\text{edit}}$ from $\mathcal{C}_{\text{orig}}$ by leveraging existing design variation models (Xu et al., 2023) to randomly modify certain parts of the original CAD model while keeping others unchanged. Next, we generate the corresponding \mathcal{I} by utilizing LVLMs to summarize the differences between the paired CAD models. Finally, we assemble these components into triplets.

Based on seq2seq formulation and triplet-correspondence dataset, we propose a locate-then-infill framework to tackle the composite nature of text-based CAD editing (Section 5). Specifically, we decompose the problem into two focused sub-tasks: locating and infilling, each handling a more manageable aspect of the overall problem. In the locating stage, we predict a masked CAD sequence $\mathcal{C}_{\text{mask}}$ by inserting special $\langle \text{mask} \rangle$ tokens into $\mathcal{C}_{\text{orig}}$, indicating the regions that require modification, i.e., $\mathcal{C}_{\text{mask}} = f_{\text{locate}}(\mathcal{I}, \mathcal{C}_{\text{orig}})$. In the infilling stage, We generate $\mathcal{C}_{\text{edit}}$ by filling in precise modifications within the masked regions, i.e., $\mathcal{C}_{\text{edit}} = f_{\text{infill}}(\mathcal{I}, \mathcal{C}_{\text{orig}}, \mathcal{C}_{\text{mask}})$. In both stages, LLMs serves as the backbone, leveraging their natural language understanding and CAD knowledge acquired during the pre-training.

4. Automated Data Synthesis Pipeline

Figure 3 illustrates our data synthesis pipeline, comprising three key steps introduced below.

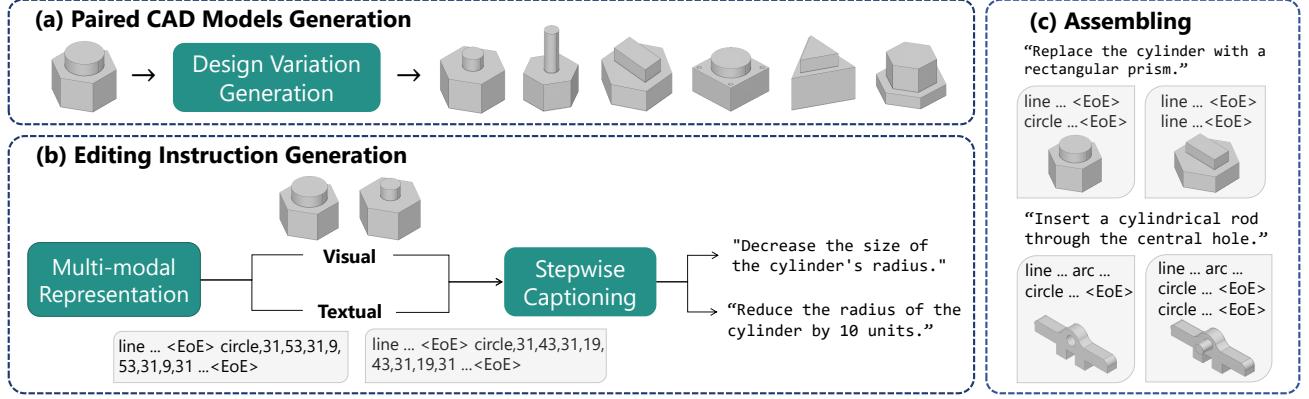


Figure 3. Illustration of automated data synthesis pipeline.

Paired CAD Models Generation. In this step, we create paired CAD models, $\mathcal{C}_{\text{orig}}$ and $\mathcal{C}_{\text{edit}}$, by starting with an existing CAD model and applying design variation models to create its variations. We source CAD models from the DeepCAD dataset (Wu et al., 2021) and use Hnc-CAD (Xu et al., 2023) as our design variation model¹. Given an existing CAD model \mathcal{C}_0 , we use Hnc-CAD’s auto-completion to generate variants $\mathcal{C}_1, \dots, \mathcal{C}_K$. We then create paired CAD models by: 1) treating \mathcal{C}_0 as the original and \mathcal{C}_k ($k \in [1, K]$) as the edited CAD model; 2) reversing the roles, considering \mathcal{C}_k as the original and \mathcal{C}_0 as the edited CAD model; and 3) using two generated variants, \mathcal{C}_{k_1} and \mathcal{C}_{k_2} ($k_1, k_2 \in [1, K], k_1 \neq k_2$), as the original and edited CAD models. This approach captures common CAD editing operations, including addition, deletion, and replacement.

Editing Instruction Generation. In this step, we generate editing instructions, \mathcal{I} , by summarizing the difference between $\mathcal{C}_{\text{orig}}$ and $\mathcal{C}_{\text{edit}}$ using LVLMs. To ensure both diversity and accuracy, we introduce two key techniques.

First, we represent CAD models in *multiple modalities*: the visual and sequence modalities. The intuition is that high-level structural changes (e.g., changing a cylinder to a cube) are more easily observed in the visual modality, while low-level numerical changes (e.g., doubling the height) are better captured in the sequence modality. Additionally, the sequence modality provides a detailed representation of all operations, ensuring that no information is lost. In our implementation, we use rendered images for the visual modality and SE operations for the sequence modality.

Second, we propose a *stepwise captioning strategy* to break down the complex task of difference summarization into three sub-tasks, enhancing generation quality of LVLMs. For each representation (visual or sequence) of a CAD

¹We choose Hnc-CAD for its well-developed open-source implementation. Other design variation models (Xu et al., 2022; Zhang et al., 2024b) are also applicable.

model pair, we follow these steps: 1) describing each CAD model – this involves analyzing geometric properties such as component types, quantities, sizes, and spatial relationships; 2) identifying differences – using the detailed descriptions from the previous step alongside CAD models, this stage extracts the necessary modifications between CAD models; and 3) compressing instructions – the final step refines the editing instructions into a concise yet precise form.

Assembling. Finally, we assemble CAD pairs ($\mathcal{C}_{\text{orig}}$ and $\mathcal{C}_{\text{edit}}$) from the first step and editing instruction (\mathcal{I}) from the second step into triplets, constructing the training dataset $\mathbb{D} = \{(\mathcal{I}, \mathcal{C}_{\text{orig}}, \mathcal{C}_{\text{edit}})\}_1^N$. Note that the visual modality is only used in the second stage to generate diverse editing instructions. In the final dataset, all CAD models are represented as SE operations, aligning with the focus of this work – text-based CAD editing in the SE domain.

5. Locate-then-Infill Framework

Figure 4 illustrates our framework. We decompose text-based CAD editing by explicitly introducing a masked CAD sequence, $\mathcal{C}_{\text{mask}}$, to indicate potential modification regions:

$$\begin{aligned} P(\mathcal{C}_{\text{edit}} | \mathcal{C}_{\text{orig}}, \mathcal{I}) &\triangleq \\ P(\mathcal{C}_{\text{mask}} | \mathcal{C}_{\text{orig}}, \mathcal{I}) \cdot P(\mathcal{C}_{\text{edit}} | \mathcal{C}_{\text{orig}}, \mathcal{I}, \mathcal{C}_{\text{mask}}), \end{aligned} \quad (1)$$

where $P(\cdot)$ denotes the probability. Here, $P(\mathcal{C}_{\text{mask}} | \mathcal{C}_{\text{orig}}, \mathcal{I})$ represents locating stage, while $P(\mathcal{C}_{\text{edit}} | \mathcal{C}_{\text{orig}}, \mathcal{I}, \mathcal{C}_{\text{mask}})$ corresponds to the infilling stage.

5.1. Locating Stage

This stage aims to generate a masked CAD sequence, $\mathcal{C}_{\text{mask}}$, where regions requiring modification are marked by special tokens $\langle \text{mask} \rangle$ while unchanged parts are copied from the original CAD sequence $\mathcal{C}_{\text{orig}}$. We adopt LLMs as the backbone and autoregressively predict tokens in $\mathcal{C}_{\text{mask}}$ using

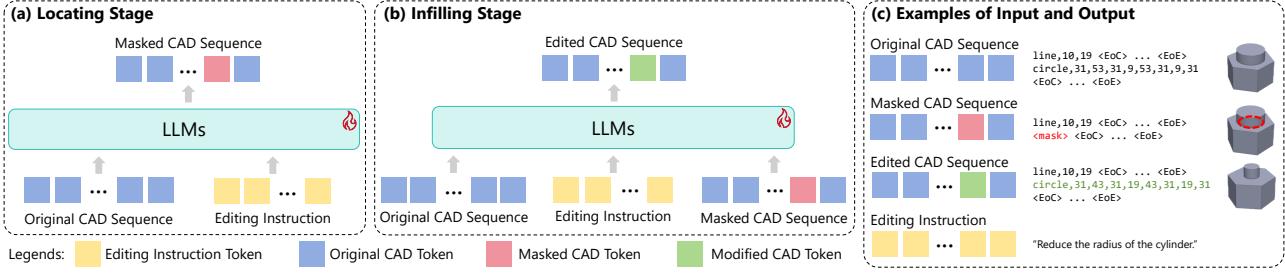


Figure 4. (a)-(b): Overview of Locate-then-Infill framework. (c): Examples of input and output, where the left column shows abstracted representations using legends, the middle column displays concrete sequences and the right column presents rendered visual objects.

$\mathcal{C}_{\text{orig}}$ and the editing instruction \mathcal{I} as context:

$$P(\mathcal{C}_{\text{mask}} \mid \mathcal{C}_{\text{orig}}, \mathcal{I}) = \prod_{t=1}^T P(c_{\text{mask}}^t \mid \mathcal{C}_{\text{orig}}, \mathcal{I}, c_{\text{mask}}^{<t}), \quad (2)$$

where T is the sequence length.

Creating Ground-Truth Masked CAD Sequence via LCS. To finetune LLMs for the locating task, we require ground-truth masked CAD sequences, denoted as $\mathcal{C}_{\text{gt-mask}}$ as supervision signals. We obtain them using the *Longest Common Subsequence* (LCS) algorithm. Let $\mathcal{C}_{\text{orig}} = \{c^1, \dots, c^T\}$ and $\mathcal{C}_{\text{edit}} = \{\tilde{c}^1, \dots, \tilde{c}^T\}$ denote the tokenized original and edited CAD sequences respectively. The LCS algorithm computes the longest subsequence $\mathcal{C}_{\text{LCS}} = \{c^{i_1}, \dots, c^{i_k}\}$ such that $\mathcal{C}_{\text{LCS}} \subseteq \mathcal{C}_{\text{orig}}$ and $\mathcal{C}_{\text{LCS}} \subseteq \mathcal{C}_{\text{edit}}$, where the indices i_1, i_2, \dots, i_k represent the positions of matching tokens in $\mathcal{C}_{\text{orig}}$. Using \mathcal{C}_{LCS} , we construct $\mathcal{C}_{\text{gt-mask}}$ as follows: 1) for each token $c^i \in \mathcal{C}_{\text{orig}}$, if $c^i \in \mathcal{C}_{\text{LCS}}$, retain c^i in $\mathcal{C}_{\text{gt-mask}}$ and if $c^i \notin \mathcal{C}_{\text{LCS}}$, replace c^i with the placeholder token `<mask>`; 2) for tokens in $\mathcal{C}_{\text{edit}}$ that do not appear in \mathcal{C}_{LCS} (representing insertions), insert `<mask>` tokens at the corresponding positions in $\mathcal{C}_{\text{gt-mask}}$; 3) consecutive `<mask>` tokens are merged into a single `<mask>` token for simplicity.

5.2. Infilling Stage

This stage focuses on generating the final edited sequence $\mathcal{C}_{\text{edit}}$ by precisely filling in the masked regions while preserving the unmodified parts. We employ LLMs as the backbone, autoregressively predicting tokens in $\mathcal{C}_{\text{edit}}$ using $\mathcal{C}_{\text{mask}}$ from the locating stage along with $\mathcal{C}_{\text{orig}}$ and \mathcal{I} as inputs:

$$\begin{aligned} & P(\mathcal{C}_{\text{edit}} \mid \mathcal{C}_{\text{mask}}, \mathcal{C}_{\text{orig}}, \mathcal{I}) \\ &= \prod_{t=1}^T P(c^t \mid \mathcal{C}_{\text{mask}}, \mathcal{C}_{\text{orig}}, \mathcal{I}, c^{<t}). \end{aligned} \quad (3)$$

Improving Performance with Selective Data. To fine-tune LLMs for the infilling task, we use the dataset \mathbb{D} synthesized through the pipeline introduced in Section 4. While

we strive to ensure high-quality synthetic data, achieving absolute accuracy is impossible. To further improve performance, we introduce a *selective dataset* curated with human annotations. Rather than having human annotators create editing instructions or CAD models from scratch, we adopt a more efficient approach — inviting them to select the best option from generated candidates. This significantly reduces human effort and accelerates dataset construction. Specifically, we first fine-tune LLMs on synthetic data and use them to generate multiple edited CAD sequences. These sequences are then rendered into visual objects, and human annotators select the best one. The chosen sequence, along with its corresponding original CAD model and textual instruction, is added to the selective dataset. Finally, we further fine-tune LLMs using this selective dataset.

5.3. Training and Inference

Training. In the locating stage, we fine-tune LLMs using Low-Rank Adapters (LoRA) (Hu et al., 2022) with the ground-truth masked CAD sequence constructed via LCS. For the infilling stage, we first fine-tune LLMs using LoRA with the synthetic dataset introduced in Section 4. We then further refine the model by fine-tuning it with LoRA on the selective dataset introduced in Section 5.2.

Inference. During inference, the locating and the infilling stage operates sequentially. The locating stage generates $\mathcal{C}_{\text{mask}}$ using $\mathcal{C}_{\text{orig}}$ and \mathcal{I} as input. The infilling stage generates $\mathcal{C}_{\text{edit}}$ by using $\mathcal{C}_{\text{mask}}$ from the locating stage along with $\mathcal{C}_{\text{orig}}$ and \mathcal{I} as input.

6. Experiments

6.1. Experimental Setup

Datasets. We use the DeepCAD dataset (Wu et al., 2021) which contains 178k CAD models. We split it into 90% training, 5% validation, and 5% testing segments. We follow the same strategy in existing work (Xu et al., 2022; 2023) to remove duplication. For the synthetic dataset used

CAD-Editor

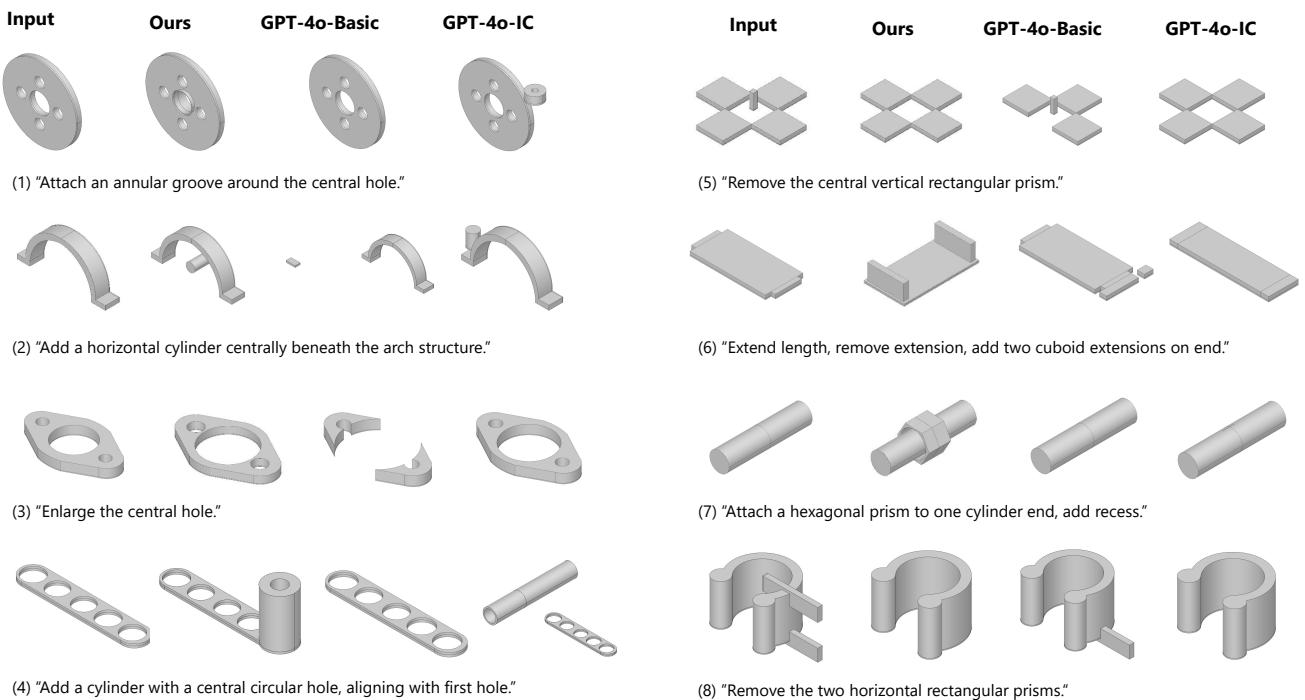


Figure 5. Qualitative results from CAD-Editor, GPT-4o-Basic and GPT-4o-IC .

for training, we generate 120k examples using the method introduced in Section 4. For the test set, we randomly sample 2k examples from the original test segment, generate the initial version following Section 4, and manually examine them to ensure the correctness. To compare the performance of different methods, we generate 5 outputs for each example in the test set, yielding 10k CAD models for evaluation.

Implementation Details. During data synthesis, GPT-4o is utilized for the visual modality, while LLaMA-3-70B handles the sequence modality. For training, we adopt Llama-3-8B-Instruct as the backbone model, fine-tuning it over 70 epochs using PyTorch’s Distributed Data Parallel (DDP) framework on 4 NVIDIA A800-80GB SMX GPUs. The initial learning rate is set to 1e-4 with a maximum token length of 1024. We employ LoRA with a rank of 32. During the inference, we set the temperature as 0.9 and top-p as 0.9 to generate varied results in each trial.

Baselines. We compare our results with three types of baselines: 1) design variation generation models, including **SkexGen** (Xu et al., 2022), **Hnc-CAD** (Xu et al., 2023) and **FlexCAD** (Zhang et al., 2024b); 2) text-based CAD generation models such as **Text2CAD** (Khan et al., 2024a) and 3) foundation models that are not specifically designed for CAD generation but have acquired some CAD knowledge during pre-training. For the third category, we select one of the most powerful foundation models, GPT-4o, as our baseline. We use two prompting strategies: 1) **GPT-4o-**

Basic, which provides only an explanation of CAD operation sequences; and 2) **GPT-4o-IC**: which includes three in-context (IC) examples retrieved based on cosine similarity between editing instructions, in addition to the basic explanation. Notably, existing CAD design variation models and text-based CAD generation models do not support text-based CAD editing. We include them as baselines to show that our model can generate CAD models with comparable or superior validity and quality while addressing a more complex task.

6.2. Metrics

As this work is the first to address text-based CAD editing, we propose evaluating the task from three key aspects.

(1) Validity. The generated CAD sequence must be valid, that is, it can be successfully parsed and rendered into a 3D visual object. We denote this as **Valid Ratio (VR)**.

(2) Realism. The generated CAD models should be realistic and similar to ground-truth CAD models. To measure this, we adopt the **Jensen-Shannon Divergence (JSD)** from prior work (Wu et al., 2021; Xu et al., 2022; 2023). JSD quantifies the similarity between two probability distributions, evaluating how often the ground-truth point clouds occupy similar positions as the generated point clouds.

(3) Edit Consistency. The generated edits should faithfully reflect the provided textual instructions. We assess

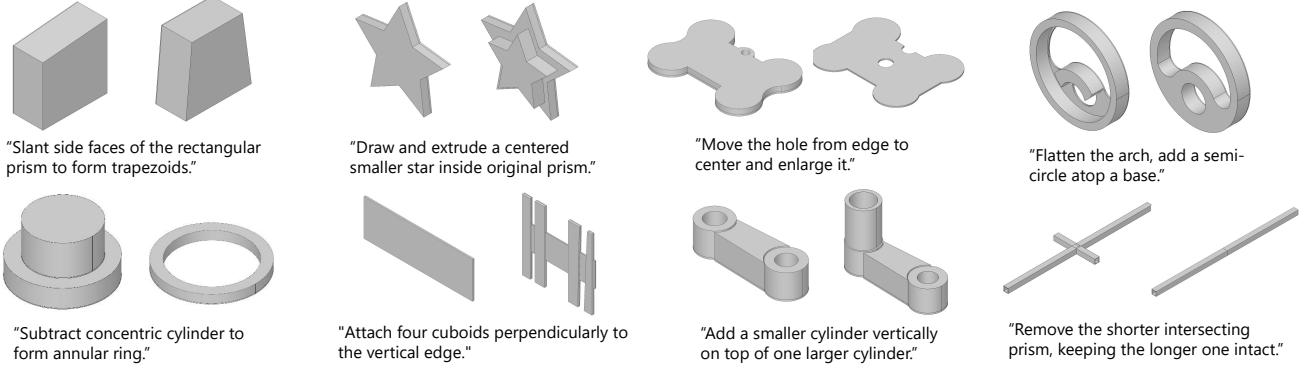


Figure 6. Additional results from CAD-Editor with various editing instructions . In each sub-figure, the left image shows the original CAD model, the right image displays the edited CAD model and the text below provides the editing instruction.

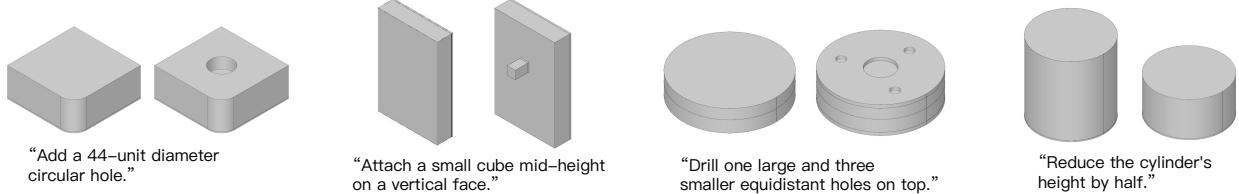


Figure 7. CAD-Editor can deal with parametric instructions, including explicit numeric expressions and implicit parametric cues.

consistency at both the point cloud and image levels:

- **Point Cloud Level.** We use the **Chamfer Distance (CD)** as a reconstruction metric as used in (Khan et al., 2024b) to evaluate geometric alignment between the edited and ground-truth CAD model pairs. While CD captures shape similarity, it may not fully reflect semantic consistency, as the goal of editing is not exact replication but meaningful transformation aligned with the instruction.

- **Image Level.** We adapt **Directional CLIP Score (D-CLIP)** from the image editing domain (Gal et al., 2022; Brooks et al., 2023) as the metric. Building upon CLIP score (Radford et al., 2021; Sohn et al., 2023), D-CLIP evaluates how well the change between the image for the edited CAD model and the image for the original CAD model aligns with the editing instruction:

$$\text{D-CLIP} = \frac{\Delta I \cdot \Delta T}{|\Delta I||\Delta T|}, \quad (4)$$

$$\Delta T = E_T(t_{\text{edit}}) - E_T(t_{\text{orig}}), \Delta I = E_I(i_{\text{edit}}) - E_I(i_{\text{orig}}),$$

where E_I and E_T are CLIP’s image and text encoders, respectively. t_{orig} is a neutral text (e.g., “This is a 3D shape.”), t_{edit} is the concatenation of t_{orig} and the textual instruction. i_{edit} and i_{orig} are the images for the edited and original CAD model, respectively.

Table 1. Quantitative evaluations. SkexGen, Hnc-CAD, FlexCAD and Text2CAD do not support text-based editing, so only their generation quality is compared. JSD, CD, and D-CLIP values are scaled by 10^2 . ↑: the higher the better, ↓: the lower the better.

Method	VR ↑	JSD ↓	CD ↓	D-CLIP ↑	H-Eval ↑
SkexGen	74.3	1.94	-	-	-
Hnc-CAD	77.4	1.77	-	-	-
FlexCAD	82.1	1.72	-	-	-
Text2CAD	84.8	2.39	1.91	-	-
GPT-4o-Basic	63.2	1.10	2.30	- 1.08	7.22
GPT-4o-IC	84.5	0.70	1.55	- 0.11	15.6
CAD-Editor	95.6	0.65	1.18	0.11	43.2

6.3. Main Results

Quantitative Evaluation. Table 1 reports the average scores across 3 runs. Notably, CAD-Editor achieves a high Valid Ratio of 95.6%, significantly surpassing other methods and indicating a greater proportion of valid and high-quality CAD generations. In terms of CD and D-CLIP, which measure alignment with editing instructions, CAD-Editor achieves scores of 1.18 and 0.11 respectively, representing substantial improvements over both GPT-4o-Basic and GPT-4o-IC. These results underscore CAD-Editor’s ef-

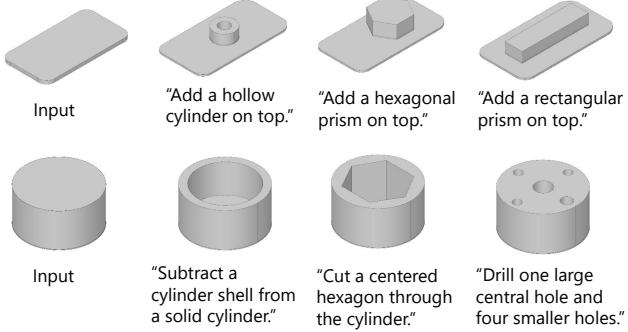


Figure 8. Given one CAD model and various instructions, CAD-Editor produces different outcomes.

effectiveness in adhering to user instructions. Additionally, CAD-Editor outperforms all baselines on the point cloud evaluation metric JSD, demonstrating good generation quality. Overall, the results indicate that CAD-Editor not only enable precise text-based CAD editing, which better alignment with user instructions, but also achieve higher validity and better quality of the generated CAD designs.

Qualitative Evaluation. In Figure 5, we qualitatively compare our method with GPT-4o-Basic and GPT-4o-IC. We observe that GPT-4o-Basic often generates irrelevant edits (case 6), unrealistic shapes (case 3), or fails to make any changes (cases 1, 4 and 7). Additionally, it struggles with distinguishing shape types (case 2) and locating specific positions (case 5). It performs better with dynamic few-shot prompting (i.e., GPT-4o-IC), highlighting the high quality of our synthetic data. GPT-4o-IC can detect specific shapes reasonably well but still struggles with precise localization (case 2 and 4) and object count (case 8). In contrast, our model successfully executes many challenging edits, including modifying sizes, shapes, and positions, as well as replacing, adding, and removing objects.

Besides, we present more results from CAD-Editor. Figure 6 shows how CAD-Editor handles a variety of editing instructions and different CAD models. Figure 7 shows that CAD-Editor has the ability to interpret parameterized instructions, including explicit numeric expressions such as “add a 44-unit diameter circular hole” and implicit parametric cues like “reduce the cylinder’s height by half”. Figure 8 illustrates that, given the same original CAD model, CAD-Editor applies different modifications based on the provided editing instructions. Figure 9 shows that when the editing instruction is vague, CAD-Editor generates multiple CAD models, all aligning with the user’s intent. Figure 10 highlights the iterative editing capability of CAD-Editor, allowing users to refine a CAD model through successive instructions until a satisfactory result is achieved.

Human Evaluation. We randomly sampled 2,000 CAD

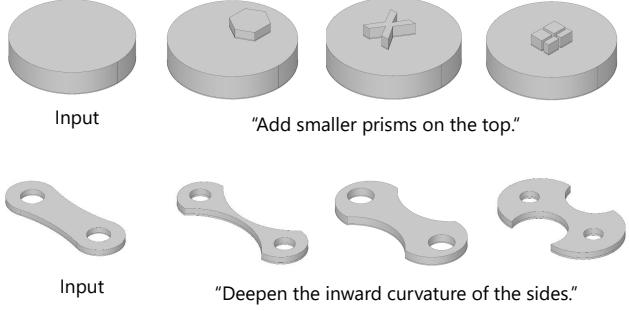


Figure 9. Given the same CAD model and instruction, CAD-Editor produces diverse outcomes.

Table 2. Ablation studies. The CAD-Editor-mini is trained on a small set with 10k examples.

Method	VR ↑	JSD ↓	CD ↓	D-CLIP ↑
CAD-Editor-mini w/ Basic	88.8	0.78	1.25	- 0.12
CAD-Editor-mini w/ Step	90.1	0.70	1.22	- 0.07
CAD-Editor w/ Direct	86.1	0.77	1.26	- 0.19
CAD-Editor w/ L-I	96.5	0.67	1.13	0.03
CAD-Editor w/ L-I & HS	95.6	0.65	1.18	0.11

models from the full set of generated results. Each pair of original CAD model and edited CAD model was independently rated by five crowd workers. For each pair, a score of 1 is assigned if the generated data is deemed successful, and 0 otherwise. Success is defined by two criteria: alignment with the text and sufficiently high visual quality. The results, denoted as **H-Eval**, are presented in Table 1. CAD-Editor outperforms baselines, indicating that crowd workers frequently found CAD models generated by baselines to be misaligned with the instructions or of lower quality, whereas our method demonstrated superior performance.

6.4. Ablation Studies

Stepwise Captioning Strategy. As introduced in Section 4, we propose a stepwise captioning strategy to decompose the complex task of generating editing instructions. To evaluate its effectiveness, we conduct an experiment where LLMs are directly queried to generate editing instructions, denoted as **CAD-Editor w/ Basic**. Our approach, which incorporates stepwise captioning, is referred to as **CAD-Editor w/ Step**. Due to resource constraints, we compare these methods on a subset of 10k examples. Table 2 shows results. CAD-Editor w/ Step outperforms CAD-Editor w/ Basic across all metrics, highlighting the importance of stepwise captioning strategy in ensuring accurate editing instruction generation.

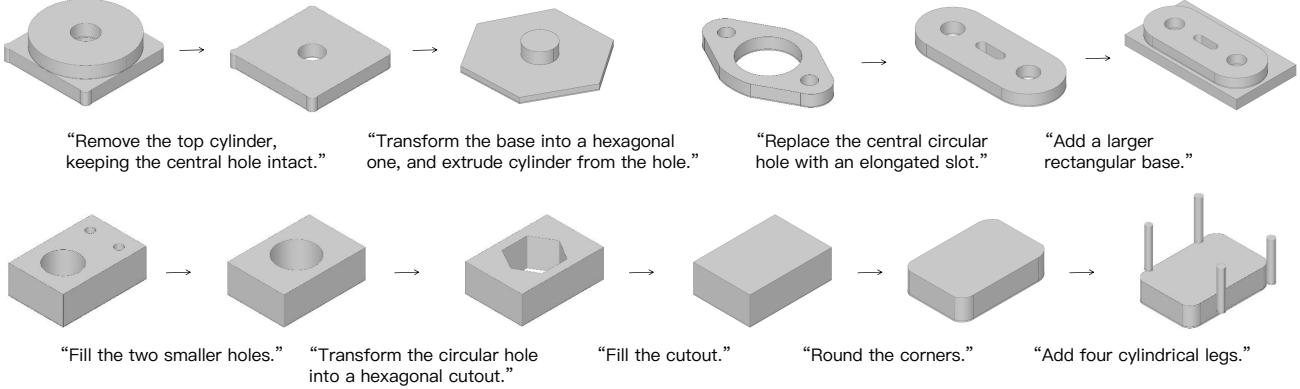


Figure 10. Apply CAD-Editor iteratively to edit a CAD model until it meets user requirements.

Locate-then-Infill Framework. Our approach consists of two stages: the locating and infilling stages, each focused on a specialized sub-task of text-based CAD editing. A more straightforward approach would be to treat the task as a whole, without decomposing it, and directly fine-tune LLMs. We refer to this as **CAD-Editor w/ Direct** and compare it with our method, **CAD-Editor w/L-I**. As shown in Table 2, CAD-Editor w/L-I outperforms CAD-Editor w/ Direct, showing the effectiveness of our two-stage approach in addressing the composite nature of text-based CAD editing.

Selective Dataset. In Section 5.2, we propose improving the performance of the infilling stage with selective data curated with human annotation. We conduct experiments both with or without such human selection, denoted as **CAD-Editor w/ L-I & HS** and **CAD-Editor w/ L-I**. Table 2 shows that this strategy results in the greatest improvement in both JSD and D-CLIP scores, demonstrating its effectiveness in enhancing generation quality and better aligning editing instructions with the edited CAD model.

7. Limitation

While CAD-Editor shows promising results, it has limitations. First, its data synthesis pipeline relies on LVLMs, which are costly and struggle with processing multiple images simultaneously. Second, as an LLM-based system, it faces challenges with long contexts and generating extended sequences, limiting its ability to handle highly complex CAD models and their corresponding edits.

8. Conclusion

We introduced CAD-Editor, the first framework for text-based CAD editing. We proposed a data synthesis pipeline to address the need of triplet data with accurate correspondence, and a locate-then-infill framework to handle the composite nature of the task. Experiments showed that CAD-

Editor outperforms other methods. In the future, we aim to enhance our data synthesis pipeline to make it more cost-effective and efficient. We also plan to develop an advanced benchmark that better reflects practical user scenarios.

Acknowledgements

We thank Ruiyu Wang for his assistance in conducting one of the baseline experiments. We also appreciate the constructive comments from the anonymous reviewers.

Impact Statement

This work presents a novel task: text-based CAD editing, along with an automated data annotation pipeline and a locate-then-infill framework to effectively address it. By streamlining the CAD model development process, our approach has the potential to significantly accelerate design workflows and make CAD modeling more accessible to a broader audience, especially those with limited design expertise. Rather than replacing human designers, our goal is to augment creativity and productivity, empowering individuals to bring their ideas to life more efficiently and effectively.

Ethics Statement

The data used in this work is tailored for creating and modifying CAD models. Due to its specialized nature, the misuse risk is naturally minimized, ensuring that the developed methods primarily benefit design and engineering tasks.

References

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

- Brooks, T., Holynski, A., and Efros, A. A. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18392–18402, 2023.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.
- Camba, J. D., Contero, M., and Company, P. Parametric cad modeling: An analysis of strategies for design reusability. *Computer-Aided Design*, 74:18–31, 2016.
- Ceylan, D., Huang, C.-H. P., and Mitra, N. J. Pix2video: Video editing using image diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 23206–23217, 2023.
- Chai, W., Guo, X., Wang, G., and Lu, Y. Stablevideo: Text-driven consistency-aware diffusion video editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 23040–23050, 2023.
- Gal, R., Patashnik, O., Maron, H., Bermano, A. H., Chechik, G., and Cohen-Or, D. Stylegan-nada: Clip-guided domain adaptation of image generators. *ACM Transactions on Graphics (TOG)*, 41(4):1–13, 2022.
- Gruver, N., Sriram, A., Madotto, A., Wilson, A. G., Zitnick, C. L., and Ulissi, Z. W. Fine-tuned language models generate stable inorganic materials as text. In *The Twelfth International Conference on Learning Representations*, 2024.
- Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- Khan, M. S., Dupont, E., Ali, S. A., Cherenkova, K., Kacem, A., and Aouada, D. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4713–4722, 2024a.
- Khan, M. S., Sinha, S., Uddin, S. T., Stricker, D., Ali, S. A., and Afzal, M. Z. Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.
- Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.
- Li, X., Song, Y., Lou, Y., and Zhou, X. Cad translator: An effective drive for text to 3d parametric computer-aided design generative modeling. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 8461–8470, 2024.
- Makatura, L., Foshey, M., Wang, B., Hähnlein, F., Ma, P., Deng, B., Tjandrasuwita, M., Spielberg, A., Owens, C. E., Chen, P. Y., et al. How can large language models help humans in design and manufacturing? *arXiv preprint arXiv:2307.14377*, 2023.
- Meng, C., Song, Y., Song, J., Wu, J., Zhu, J.-Y., and Ermon, S. Sdedit: Image synthesis and editing with stochastic differential equations. *arXiv preprint arXiv:2108.01073*, 2021.
- Mikaeili, A., Perel, O., Safaei, M., Cohen-Or, D., and Mahdavi-Amiri, A. Sked: Sketch-guided text-based 3d editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14607–14619, 2023.
- OpenAI. Introducing chatgpt. OpenAI Blog, 2023. Available: <https://openai.com/blog/chatgpt>.
- Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763, 2021.
- Shahin, T. M. Feature-based design—an overview. *Computer-Aided Design and Applications*, 5(5):639–653, 2008.
- Sohn, K., Ruiz, N., Lee, K., Chin, D. C., Blok, I., Chang, H., Barber, J., Jiang, L., Entis, G., Li, Y., et al. Styledrop: text-to-image generation in any style. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pp. 66860–66889, 2023.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022a.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837, 2022b.

- Willis, K. D., Pu, Y., Luo, J., Chu, H., Du, T., Lambourne, J. G., Solar-Lezama, A., and Matusik, W. 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.
- Wu, R., Xiao, C., and Zheng, C. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6772–6782, 2021.
- Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., Lin, Q., and Jiang, D. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024.
- Xu, X., Willis, K. D., Lambourne, J. G., Cheng, C.-Y., Jayaraman, P. K., and Furukawa, Y. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. In *International Conference on Machine Learning*, pp. 24698–24724, 2022.
- Xu, X., Jayaraman, P. K., Lambourne, J. G., Willis, K. D., and Furukawa, Y. Hierarchical neural coding for controllable cad model generation. In *International Conference on Machine Learning*, pp. 38443–38461, 2023.
- Yu, L., Jiang, W., Shi, H., Jincheng, Y., Liu, Z., Zhang, Y., Kwok, J., Li, Z., Weller, A., and Liu, W. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.
- Yue, L., Liu, Q., Zhao, L., Wang, L., Gao, W., and An, Y. Event grounded criminal court view generation with cooperative (large) language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2221–2230, 2024.
- Zhan, Y., Liu, Q., Gao, W., Zhang, Z., Wang, T., Shen, S., Lu, J., and Huang, Z. Coderagent: Simulating student behavior for personalized programming learning with large language models, 2025.
- Zhang, Y., Huang, D., Liu, B., Tang, S., Lu, Y., Chen, L., Bai, L., Chu, Q., Yu, N., and Ouyang, W. Motiongpt: Finetuned llms are general-purpose motion generators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 7368–7376, 2024a.
- Zhang, Z., Sun, S., Wang, W., Cai, D., and Bian, J. Flexcad: Unified and versatile controllable cad generation with fine-tuned large language models. *arXiv preprint arXiv:2411.05823*, 2024b.
- Zhao, L., Wang, Y., Liu, Q., Wang, M., Chen, W., Sheng, Z., and Wang, S. Evaluating large language models through role-guide and self-reflection: A comparative study. In *The Thirteenth International Conference on Learning Representations*, 2025.

A. CAD-Editor Dataset Preprocessing

The DeepCAD dataset exhibits a severe data imbalance. Among deduplicated 137,012 training models, over 91.1% have three or fewer SE operations, while only 6.2% have 4–5 SEs. Models with more than 5 SEs make up less than 3%, and those exceeding 10 SEs account for just 0.4%. To mitigate the impact of this long-tail distribution, we limit our dataset to models with at most 3 SE pairs.

Additionally, the design variation generation model can sometimes introduce noise by altering a CAD model too drastically or not at all. To ensure data quality, we implement a data filtering strategy. We filter out CAD pairs with too many changes by excluding examples with more than three editing instructions or more than five `<mask>` tokens. We also filter out pairs with no significant changes by excluding instructions containing phrases like “no transformation is needed”.

B. Stepwise Captioning Strategy

To implement stepwise captioning, we utilize GPT-4o four times per CAD pair to generate the final image-level editing instructions. For sequence-level, we employ LLaMA-3-70B. The detailed prompts are illustrated in Figure 11.

Visual Modality

Step 1:
Please take a look at the first of two 3D shapes we'll be examining. Please provide a detailed description, focusing on its geometric properties, including the type and number of elements it features, the proportions of its size, its positional relationships between elements, and any additional details that stand out.

Step 2:
Now, let's turn our attention to the second 3D shape. Please provide a detailed description, focusing on its geometric properties, including the type and number of elements it features, the proportions of its size, its positional relationships between elements, and any additional details that stand out.

Step 3:
Please provide detailed instructions for transforming the first 3D shape into the second.

Step 4:
Condense your instructions to one sentence, 10 words maximum.

Sequence Modality

Task
You are a senior Computer-Aided Design (CAD) engineer. Your task is to provide a clear and concise editing instruction (10 words or fewer) for editing a sketch-and-extrude CAD model. Your response should include:

1. Description of the Original CAD Model: Analyze the CAD operation sequence and describe the resulting geometry. Include element types (e.g., cylinder, prism, hole), quantities, proportions, spatial relationships, and any notable details.

2. Description of the Edited CAD Model: Analyze the CAD operation sequence and describe the resulting geometry. Include element types (e.g., cylinder, prism, hole), quantities, proportions, spatial relationships, and any notable details.

3. Change Analysis:

- Geometric Changes: Describe added, removed, or modified elements, including types (e.g., cylinder, prism, hole) and quantities (e.g., two rectangles). Use spatial or geometric features (e.g., "upper triangular face", "smaller rectangular prism", "central circular hole") instead of unintuitive terms like "first" or "second."
- Proportions and Dimensions: Note changes in size, scaling, or relative proportions.
- Positional Relationships: Explain spatial alignment and relationships between elements.
- Other Notable Details: Highlight any additional observations.
- Purpose: Suggest the intent behind the edit (e.g., "add a central hole", "remove the smaller prism", or "increase length by 8 units").

4. Editing Instruction: Provide a concise instruction (max 10 words) describing the modification.

Sketch-and-Extrude Model Overview
...

Your Task
Original CAD Sequence: [original CAD sequence]
Edited CAD Sequence: [edited CAD sequence]
Let's think step by step. Your output should be of the following json format:
{
 "Description of the Original CAD Model": your description here.
 "Description of the Edited CAD Model": your description here.
 "Change Analysis": your change analysis here.
 "Editing Instruction": the final editing instruction here (10 words maximum).
}

Figure 11. Detailed prompt used for stepwise captioning.

Furthermore, Figure 12 presents a comparison between the basic captioning approach and our proposed stepwise captioning method.

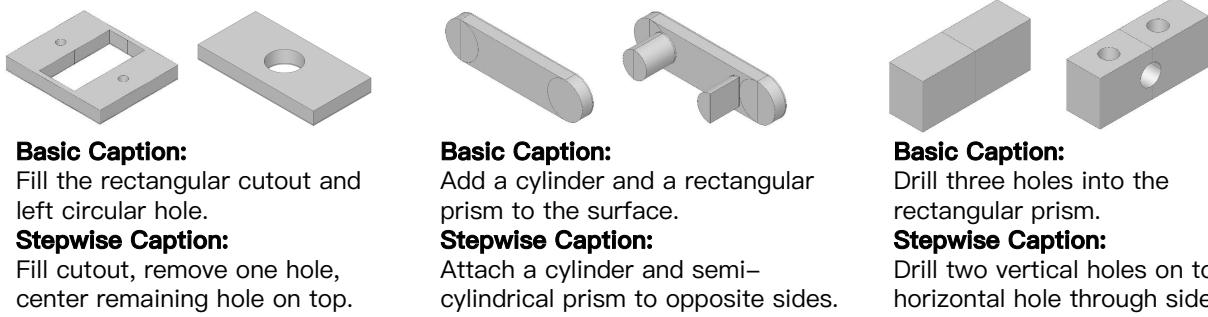


Figure 12. Comparison between the basic captioning method and our stepwise captioning method.

C. Locate-then-Infill Framework

We adopt task-specific prompts for both the locating and infilling stages. The detailed prompt designs for each task are shown in Table 3.

Table 3. Prompt for Locate-then-Infill framework.

Locating Prompt	Infilling Prompt
<p>Below is a Computer-Aided Design (CAD) operation sequence. Replace the parts that need to be modified with the string “<mask>” according to the editing instruction.</p> <p>Original CAD Operation Sequence: [Original CAD sequence]</p> <p>Editing Instruction: [Textual editing instruction]</p> <p>Masked CAD Operation Sequence: [Masked CAD sequence]</p>	<p>Below is the original Computer-Aided Design (CAD) operation sequence.</p> <p>Original CAD Operation Sequence: [Original CAD sequence]</p> <p>The parts that need to be modified according to the editing instruction have been replaced by the string “<mask>”.</p> <p>Editing Instruction: [Textual editing instruction]</p> <p>Masked CAD Operation Sequence: [CAD sequence with “<mask>”]</p> <p>Generate the edited CAD sequence that could replace “<mask>” in the CAD model:</p>

D. Additional Qualitative Results

We present qualitative comparisons between the directly fine-tuned LLM and our proposed Locate-then-Infill framework in Figure 13. Compared to the direct fine-tuning approach, our framework improves generation quality, enhances text-CAD alignment, and ensures greater output stability.

Additional qualitative comparisons between CAD-Editor and other baseline methods are shown in Figure 14, demonstrating the superior performance of CAD-Editor under various editing conditions.

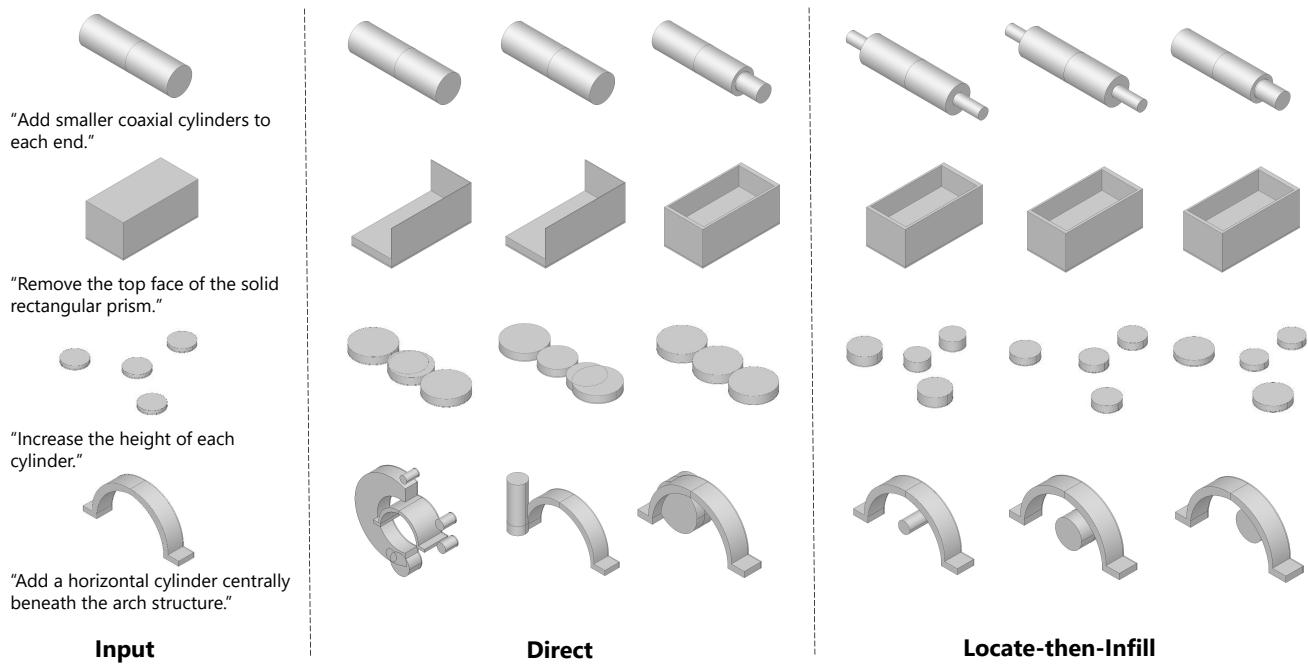


Figure 13. The qualitative comparison between the directly fine-tuned LLM and our Locate-then-Infill framework.

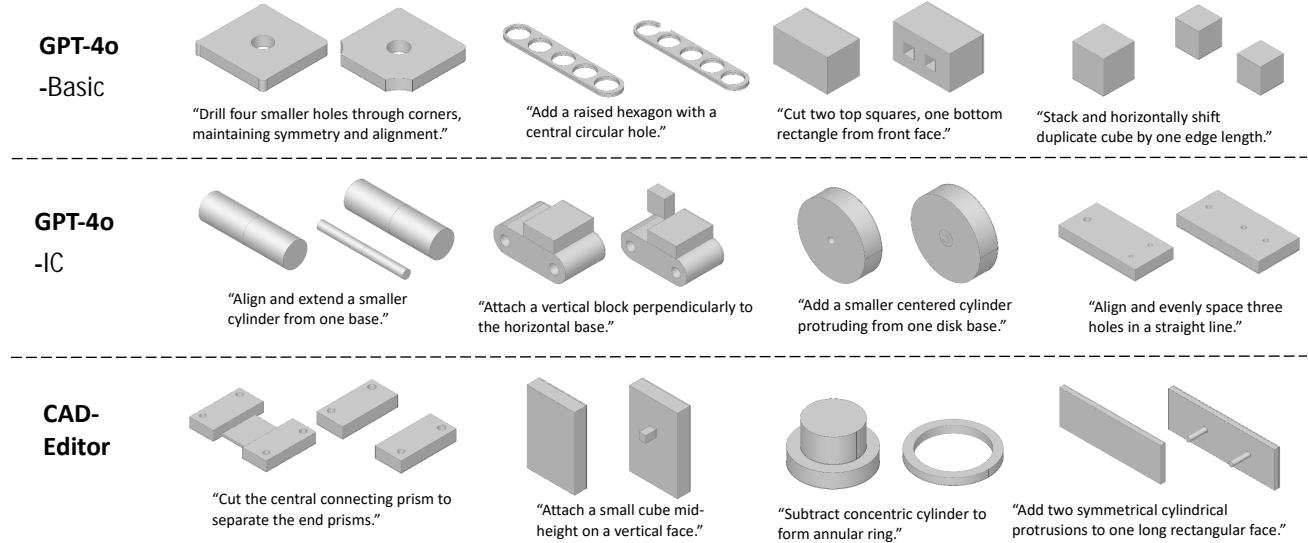


Figure 14. Additional qualitative comparison results between CAD-Editor and baseline methods.

E. Scaling & Generalization

Due to the severe data imbalance in DeepCAD, we constrain the SE length during training. To evaluate the scalability of our method, we conduct an additional experiment in which we artificially balance the dataset across different SE lengths. Specifically, we construct a training set containing 4,000 samples for each SE length from 1 to 5, ensuring a uniform distribution. The model is then evaluated on separate test sets corresponding to each SE length. As shown in Table 4, CAD-Editor consistently outperforms the baselines across all SE lengths. Model performance declines slightly as SE length increases. Since generating longer sequences is inherently more challenging, this result demonstrates that our method generalizes well to complex CAD structures given sufficient training data.

To further assess generalization, we evaluate CAD-Editor’s generalization by testing a model trained on DeepCAD directly on Fusion 360 dataset (Willis et al., 2021). As shown in Table 5 , CAD-Editor outperforms baselines, confirming its generalization ability to datasets with different shape distributions.

We present additional results generated by CAD-Editor in Figure 15, including cases with more than three SEs and examples sourced from the Fusion 360 dataset.

Table 4. Quantitative evaluation across different numbers of sketch-extrude (SE) operations. We construct a dataset of 20,000 examples with a uniform distribution over SE counts from 1 to 5. JSD and D-CLIP values are scaled by 10^2 . \uparrow : higher is better, \downarrow : lower is better.

Method	JSD \downarrow					D-CLIP \uparrow					VR \uparrow				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
GPT-4o-Basic	5.98	2.86	3.68	3.82	3.95	-0.31	-0.90	-0.61	-0.47	-0.97	56.5	64.2	59.4	61.5	58.3
GPT-4o-IC	5.42	1.75	2.29	2.40	3.67	0.39	-0.06	-0.50	-0.11	-0.38	76.2	85.0	76.5	75.5	73.1
CAD-Editor	4.08	1.46	2.01	2.35	3.15	0.41	-0.23	-0.06	-0.02	-0.29	84.3	91.6	87.5	79.3	79.5

Table 5. Quantitative evaluation on the dataset generated from the Fusion 360 Gallery. Lower JSD and higher D-CLIP and VR values indicate better performance. The results demonstrate that our method exhibits strong generalization ability.

Method	JSD \downarrow	D-CLIP \uparrow	VR \uparrow
GPT4o-Basic	4.58	-0.45	57.7
GPT4o-IC	2.59	-0.39	81.5
CAD-Editor	2.34	0.41	93.9

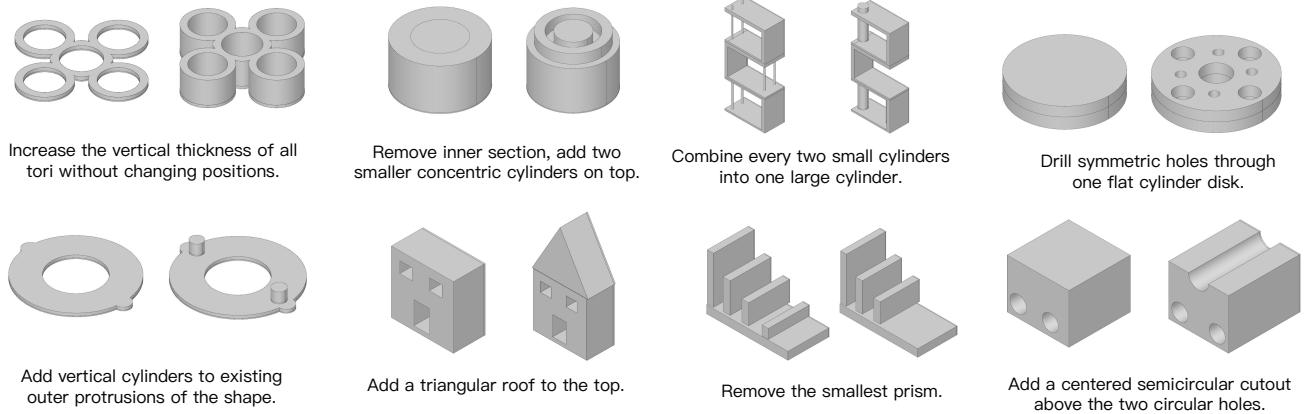


Figure 15. Extended results from CAD-Editor on more complex scenarios, including examples with more than three SEs and examples sourced from Fusion 360 datasets.

F. Baseline Editing Methods

This section presents the detailed prompt used in the GPT-4o-IC baseline, as illustrated in Figure 16. The GPT-4o-Basic prompt follows the same format, excluding the in-context examples.

GPT-4o-IC

Modify the original Computer-Aided Design(CAD) operation sequence according to the instruction:

Instructions for sketch-and-extrude model

A sketch-and-extrude model consists of multiple extruded-sketches.

Sketch

- A sketch consists of multiple faces
- A face consists of multiple loops.
- A loop consists of multiple curves.
- A curve is either a line, an arc, or a circle.
- A circle is defined by four points with four geometry tokens.
- An arc is defined by three points but with two tokens, where the third point is specified by the next curve (or the first curve when a loop is closed).
- A line is defined by start point.
- A point is represented by two integers which stands for the x and y coordinate, respectively.
- A loop with a circle can not contain additional curves since it is already a closed path.
- When a face consists of multiple loops, the first loop defines the external boundary, and the remaining loops define internal loops (i.e., holes).
- An end-primitive token appears at the end of each primitive (curve, line, face, loop or sketch).

Extrude

Each sketch will be followed by an extrude, which is represented by 18 parameters: BVVTTTTRRRRRRRRRRSOO.

- B represents one of the three Boolean operations: add, cut or intersect. It occupies 1 parameter
- V indicates the displacements of the top and the bottom planes from the reference plane in which a sketch is extruded to form a solid. It occupies 2 parameters.
- T represents 3D translation applied to the extruded solid. It occupies 3parameters.
- R represents 3D rotation of the extrusion direction. It occupies 6 parameters.
- S represents the uniform scaling factor. It occupies 1 parameter.
- O represents the center of scaling as a 2D coordinate. It occupies 2 parameters.

Note

- Note that every number is an integer.

Examples for editing sketch-and-extrude model

[The 3 most similar editing instructions and their corresponding CAD pairs]

Your task

Original CAD Command Sequence: [original sequence]

Instruction: [editing instruction]

Your output should be of the following json format:

```
{
  "edited sequence": your modified CAD sequence here.
}
```

Figure 16. Prompt employed in baseline methods for CAD model editing.