CAD-CODER: AN OPEN-SOURCE VISION-LANGUAGE MODEL FOR COMPUTER-AIDED DESIGN CODE GENERATION

Anna C. Doris[1,*], **Md Ferdous Alam**[1], **Amin Heyrani Nobari**[1], **Faez Ahmed**[1]

[1]Massachusetts Institute of Technology, Cambridge, MA

ABSTRACT

Efficient creation of accurate and editable 3D CAD models is critical in engineering design, significantly impacting cost and time-to-market in product innovation. Current manual workflows remain highly time-consuming and demand extensive user expertise. While recent developments in AI-driven CAD generation show promise, existing models are limited by incomplete representations of CAD operations, inability to generalize to real-world images, and low output accuracy. This paper introduces CAD-Coder, an open-source Vision-Language Model (VLM) explicitly fine-tuned to generate editable CAD code (CadQuery Python) directly from visual input. Leveraging a novel dataset that we created—GenCAD-Code, consisting of over 163k CAD-model image and code pairs—CAD-Coder outperforms state-of-the-art VLM baselines such as GPT-4.5 and Qwen2.5-VL-72B, achieving a 100% valid syntax rate and the highest accuracy in 3D solid similarity. Notably, our VLM demonstrates some signs of generalizability, successfully generating CAD code from real-world images and executing CAD operations unseen during fine-tuning. The performance and adaptability of CAD-Coder highlights the potential of VLMs fine-tuned on code to streamline CAD workflows for engineers and designers. CAD-Coder is publicly available at: https://github.com/anniedoris/CAD-Coder.

NOMENCLATURE

LLM	Large Language Model
VLM	Vision-Language Model

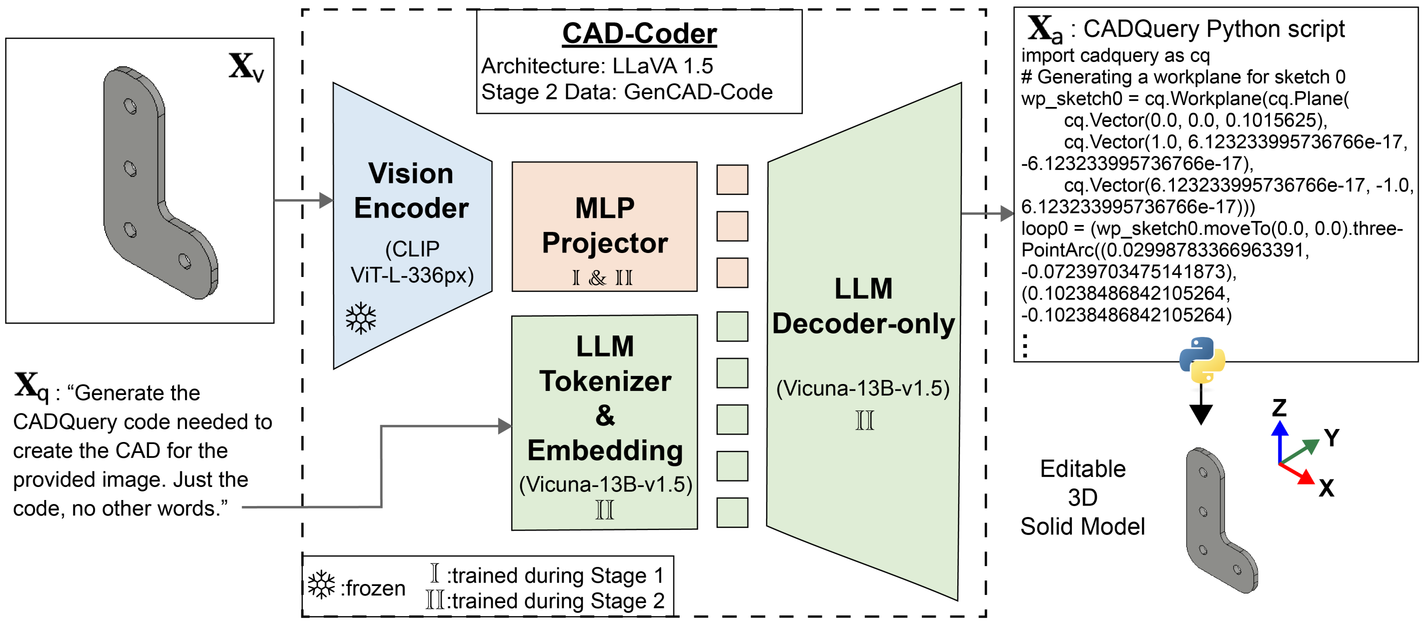
1. INTRODUCTION

Generative AI has recently shown great promise in engineering design, with large language models (LLMs) and vision-language models (VLMs) beginning to automate parts of the engineering design process [1]. An indispensable step in the design of any engineering system, consumer product, or structure is the creation of a 3D Computer-Aided Design (CAD) model. With the exception of specific functionalities, like generative design and topology optimization, the process of CAD modeling – drawing sketches, defining constraints, extruding bodies – remains largely manual, performed by designers with years of engineering experience and familiarity with CAD modeling software. This has motivated researchers to seek ways to partially automate CAD creation using learning-based approaches. An AI assistant capable of generating editable CAD models from textual or visual specifications would significantly reduce the time engineers spend on manual tasks. It would also make CAD modeling more accessible to beginners. An important goal of the research has been to develop generative models that can produce valid, parametric CAD models from high-level inputs, thereby accelerating the design-to-production pipeline. Unlike most 3D shape generation work that outputs meshes or voxels, a CAD automation system should provide *modifiable* designs in a programmatic form, since practical engineering applications demand editability and reusability of the 3D CAD.

Early approaches to AI-driven CAD generation have made progress but face notable limitations. Many prior works trained bespoke models solely on CAD data or limited synthetic distributions, which restricted their scope of understanding. A number of works in recent years have explicitly trained ML models from scratch to generate CAD, many focusing non-editable 3D solid generation [2–6] with fewer focusing on editable, parametric CAD program generation [7–9]. Many of these past editable CAD systems focused on a small set of primitive operations, which limited their ability to handle real-world variation. CAD-specific models that are trained from scratch lack the broad visual and contextual knowledge that modern foundation models possess. As a result, prior trained-from-scratch CAD generators face two generalizability issues: 1) they may fail on inputs that deviate from their training distribution and 2) they fail when asked to use CAD operations not seen during training. As many existing AI solutions for 3D modeling either produce non-editable outputs (e.g., meshes) or rely on specialized training with limited generalization capacity, there is a need for a more robust approach to editable CAD generation.

Meanwhile, vision-language foundation models (VLMs) have achieved remarkable generalization across tasks in computer vision and natural language processing (NLP). VLMs, large language models (LLMs) integrated with vision encoders—such as

*Corresponding author: adoris@mit.edu


X_v
CAD-Coder
Architecture: LLaVA 1.5
Stage 2 Data: GenCAD-Code
Vision Encoder
(CLIP ViT-L-336px)
MLP Projector
I & II
LLM Tokenizer & Embedding
(Vicuna-13B-v1.5)
II
LLM Decoder-only
(Vicuna-13B-v1.5)
II
X_q : "Generate the CADQuery code needed to create the CAD for the provided image. Just the code, no other words."
:frozen
I :trained during Stage 1
II:trained during Stage 2
X_a : CADQuery Python script
Editable 3D Solid Model
Z
Y
X

```
import cadquery as cq
# Generating a workplane for sketch 0
wp_sketch0 = cq.Workplane(cq.Plane(
    cq.Vector(0.0, 0.0, 0.1015625),
    cq.Vector(1.0, 6.123233995736766e-17,
-6.123233995736766e-17),
    cq.Vector(6.123233995736766e-17, -1.0,
6.123233995736766e-17)))
loop0 = (wp_sketch0.moveTo(0.0, 0.0).three-
PointArc((0.02998783366963391,
-0.07239703475141873),
(0.10238486842105264,
-0.10238486842105264)
⋮
```

FIGURE 1: Overview of CAD-Coder. The VLM accepts an image as input and outputs CadQuery code, which can be run as a Python script to produce an editable, 3D solid CAD model. CAD-Coder has a LLaVA 1.5-type architecture and is fine-tuned on the GenCAD-Code dataset.

GPT-4o [1] (closed-source) or LLaVA [10] (open-source)—have the potential to mitigate these two issues. Pre-trained on billions of tokens, LLMs are already capable of generating code and therefore have the capacity to output a complete, editable CAD representation: code for CAD, using an existing, well-defined CAD scripting library. Pre-trained on millions of images, VLMs have learned meaningful latent representations of real-world images, positioning them well for CAD generation conditioned on images of real objects.

Prior studies have used prompt engineering or few-shot strategies to coax general VLMs into image-conditioned CAD generation tasks, but without domain-specific training, their performance and reliability remains limited (e.g., they may hallucinate incorrect geometry or produce code with syntax errors [11]). Fine-tuning has been used to significantly improve the performance of LLMs for text-conditioned CAD code generation [12] and for image-conditioned generation of CAD programs expressed in domain-specific languages (DSLs) [13, 14]. However, reliance on DSL-based CAD programs–rather than established, full-featured CAD scripting libraries–restricts CAD generation to the narrow set of operations defined in the DSL and reduces a model's ability to leverage its pretraining knowledge.

This raises the question: can we harness a generalist VLM to generate *parametric CAD code* conditioned on images, bridging the gap between high-level visual input and precise engineering code? To our knowledge, no published work has fine-tuned an open-source VLM specifically for the direct synthesis of CAD code using a widely-adopted scripting library. There is a gap in the literature for an approach that combines the broad CAD scripting and visual pre-training knowledge of foundation models with dedicated training on CAD code to achieve both versatility and accuracy in image-conditioned CAD scripting automation.

In this paper, we propose **CAD-Coder**, an open-source vision-language model tailored for computer-aided design code generation. We utilize a powerful, general VLM and fine-tune it end-to-end for the task of producing CAD code conditioned on image inputs. Concretely, CAD-Coder takes in an image and generates readily-executable code using CadQuery, a Python-based parametric CAD scripting library. We validate CAD-Coder through extensive experiments, demonstrating significant performance gains over the state-of-art, image-conditioned, CAD code-generating baselines. Notably, CAD-Coder attains an 100% valid syntax rate, meaning every generated CAD script compiles successfully — a substantial improvement over the compile rates of existing solutions. In terms of generated 3D solid accuracy, CAD-Coder's generated shapes closely match the ground-truth solids, improving upon the precision achieved by state-of-the-art baselines. We also conduct tests to probe generalization: for example, we feed CAD-Coder photographs of real objects (outside the fine-tuning dataset distribution) and prompt it to produce CAD code, or, we ask it to use CAD operations not seen during fine-tuning. In these challenging scenarios, CAD-Coder shows encouraging performance, often producing reasonable CAD where trained-from-scratch or DSL-based solutions would struggle.

In summary, our key contributions are as follows:

1. **A VLM Trained for CAD Code Generation:** We introduce CAD-Coder, a fine-tuned, open-source vision-language foundation model designed specifically for the generation of CAD code. A LLaVA-derived VLM, our model, given im-

[1] https://openai.com/index/gpt-4o-system-card/

age inputs, generates readily-executable Python CadQuery code. Our fine-tuned VLM approach is different from existing methods in that it is an image-conditioned method that outputs CAD in an *established code representation* (i.e. CadQuery), addressing generalization limitations of prior works.

2. **Performance Comparison**: CAD-Coder achieves superior results on the CAD code generation task, outperforming strong VLM baselines including GPT-4.5 and Qwen2.5-VL-72B. It produces valid code in 100% of cases and significantly improves geometric accuracy of generated solids over previous methods, affirming our method of fine-tuning a foundation model for this task.

3. **CAD-Coder Generalization Experiments**: We provide evidence that CAD-Coder has generalization ability to tasks not explicitly included in the model's fine-tuning dataset, owing to its inherited, broad knowledge. On a small test dataset of images of real, 3D-printed objects, we illustrate that CAD-Coder generates reasonable CadQuery code, an advantage over trained-from-scratch CAD generation approaches which struggle with images out-of-distribution with respect to their training data. We also illustrate that with careful prompting and selection of the pre-trained LLM that is fine-tuned, CAD-Coder variants can utilize CAD operations not explicitly included in the fine-tuning dataset, a benefit over existing trained-from-scratch approaches and methods reliant on DSL-based CAD programs.

4. **A Publicly Available Dataset of CAD Code Paired with Images**: We generate CadQuery Python code for 163,671 CAD models. These Python files accompany the rendered CAD images from the GenCAD [9] dataset, comprising the largest publicly available dataset, named GenCAD-Code, of CAD code paired with CAD images.

Through these contributions, our work positions foundation models as a promising avenue for next-generation CAD tools. CAD-Coder not only bridges the gap between high-level visual understanding and low-level CAD programming, but also underscores the value of marrying general VLMs with domain-specific fine-tuning to meet the requirements of engineering design. The following sections detail the methodology of CAD-Coder, its experimental evaluation against existing baselines, and a discussion of CAD-Coder's generalizability to tasks not seen during fine-tuning.

2. RELATED WORK

Since a number of works – particularly in recent years – have developed AI-based solutions for CAD generation, we narrow our focus to those existing methods that support a conditional input and generate *editable* CAD representations as output (Table 1).

2.1 Editable CAD Generation: Models & Training

The majority of the image and/or text conditioned editable-CAD generation models presented in Table 1 make use of transformers [21], which enable prediction of the next CAD operation based on the previously predicted operations. Model architectures differ in accordance with whether the model was trained from scratch or makes use of pre-trained LLMs and/or VLMs.

2.1.1 Training from Scratch. Several works [9, 15, 16] have trained conditioned, editable CAD generating models from scratch. For example, [9] trained an image-to-CAD-program model – consisting of three components – from scratch. They first trained a transformer-based encoder-decoder to learn the latent representation of the CAD commands and to autoregressively generate CAD commands. Using the frozen CAD command encoder, they contrastively trained an image encoder to learn a joint image-CAD command latent space. Finally, they train a diffusion prior model to generate CAD command latent vectors conditioned on image latent vectors. Sequentially combining the contrastively trained image encoder with the diffusion prior model with the autoregressively trained transformer decoder yields their model, GenCAD, which can generate a CAD command sequence given an input image. GenCAD achieved SOTA – outperforming DeepCAD [8] and SkexGen [7] – on chamfer-distance-based solid accuracy metrics.

2.1.2 Leveraging Foundation Models. Instead of training a model from scratch, researchers have begun utilizing LLMs and VLMs as a foundation for CAD generation and design automation. A key advantage of LLMs is their ability to handle free-form user inputs and generalize across tasks, making them attractive for engineering design applications.

Without Additional Training: A number of works have explored methods for using off-the-shelf pre-trained foundation models, including leveraging multiple models [17, 18] or integrating the models with external tools [12, 19]. Alrashedy et al. [17] argue that a single-pass LLM often fails to meet all design requirements, so they incorporate a VLM in the loop to iteratively verify and correct the output. In their CADCodeVerify framework, GPT-4 first writes CAD code given a prompt, then a VLM analyzes the rendered 3D solid and asks itself targeted questions (e.g., "Are the holes the correct diameter?"). The VLM's answers help identify deviations, which are fed back to GPT-4 to refine the code. This self-critique loop improved the shape accuracy and success rates of generated models. Another work [18] models the design process as a team of specialized AI agents and realize a human-AI co-design loop with multiple LLM-based agents taking on different roles (e.g., requirement engineer, CAD engineer, quality checker). This multi-agent approach mirrors how human design teams iterate on CAD, and was shown to produce more complete and error-checked CAD outputs than a zero-shot VLM.

CAD-Assistant by Li et al. [19] integrate a VLM "planner" with a Python-API-driven CAD engine (FreeCAD) to enable zero-shot solving of generic CAD tasks. [12] combined GPT-4 and GPT-4V with a debugger so that model-generated programs with syntax errors could be iteratively improved. Despite the power and extensive pre-training of OpenAI's GPT models, GPT-4 and GPT-4V were in many cases incapable of generating accurate CAD code. CAD model accuracy (IOU) dropped to near zero for mechanical spring CAD models, even when GPT-4 was integrated with the debugger.

TABLE 1: Overview of some existing approaches for conditional, editable-CAD generation.

Method	Conditional Input	Output: Editable-CAD Representation	2D or 3D CAD	Model Architecture	Training Method
GenCAD [9]	Image	CAD program	3D	Transformer encoder/decoder + Contrastive image encoder + Diffusion prior	From scratch
CSGNet [15]	Image	CAD program	2D/3D	Encoder (CNN) Decoder (RNN)	From scratch
Text2CAD [16]	Text	CAD program	3D	Transformer	From scratch
CADCodeVerify [17]	Text	CAD Code: CadQuery	3D	GPT-4 Gemini 1.5 Pro CodeLlama	None: Multi-Model
IdeaToCAD [18]	Image + Text	CAD Code: CadQuery	3D	GPT-4o	None: Multi-Agent
CAD-Assistant [19]	Image + Text	CAD Code: FreeCAD	3D	GPT-4o	None: Tool Augmented
LLM4CAD [11]	Image + Text	CAD Code: CadQuery	3D	GPT-4/GPT-4V + Debugger	None: Debugger
BlenderLLM [12]	Text	CAD Code: Blender	3D	Qwen2.5-Coder-7B-Instruct	Fine-tuning + Self-improvement
Text2CAD [20]	Text	CAD Code: CadQuery	3D	GPT-3.5 based	Fine-tuning
Cad-VLM [13]	Image + Text	CAD program	2D	Asymmetric transformer encoder-decoder	Fine-tuning
OpenECAD [14]	Image + Text	CAD program	3D	TinyLLaVA	Fine-tuning
CAD-Coder (Ours)	Image + Text	CAD Code: CadQuery	3D	LLaVA 1.5	Fine-tuning

With Fine-Tuning: The previously highlighted studies underscore the growing role of foundation models in CAD and engineering design. However, without additional training, there is a limit to how much a model's performance can improve on the CAD generation task. Fine-tuning, where all or some portion of a model's weights are further trained on the desired task, offers a middle-ground between training from scratch and leveraging off-the-shelf pre-trained models. Relatively few studies—[12], [20], [13], and [14]—have investigated foundation models' capacity to be fine-tuned for the CAD generation task. [12] perform end-to-end supervised fine-tuning on an LLM to improve its ability to generate Blender Python code. Finetuning results in 2X improvement in the accuracy of generated 3D solids and in a substantial reduction (~30%) in the number of generated scripts with syntax errors. Their fine-tuned model also substantially outperforms GPT-4o and other closed-source LLMs in its ability to generate Blender CAD code. Similarly, [20] fine-tunes GPT-3.5 on a specialized text-to-CadQuery dataset, significantly improving success rates of code generation. Extending to the visual domain, [13] fine-tune a pre-trained vision transformer on a dataset of sketches to improve its ability to generate 2D CAD. Yuan et al. [14] use LoRA to train OpenECAD, a fine-tuned TinyLLaVA model that outputs CAD programs given an image input, showing that their method outperforms gpt-4o-mini on an accuracy-based scoring function. The authors also illustrate the generalization potential of fine-tuned VLMs, by demonstrating that OpenECAD can generate a CAD program when conditioned on a hand-drawn sketch (despite the fine-tuning dataset only including images of CAD.)

A fine-tuned foundation model could offer several advantages over training a CAD generation model from scratch. Pre-trained models have already learned skills that are translatable to the CAD generation task. When an existing, code-based CAD representation is used (e.g. CadQuery), fine-tuning a pre-trained LLM that has pre-existing knowledge of that code could enable the model to leverage pre-training knowledge and utilize CAD operations that may have not been present in a limited fine-tuning dataset. Likewise, if image-conditioned CAD generation is desired, utilizing a pre-trained image encoder (e.g. CLIP [22]) that has been trained on millions of image-text pairs could improve the model's ability to generate a meaningful image latent vector. Even more critically, pre-trained image encoders – which have been trained on images of real-world objects – can promote CAD generation conditioned on images of *real-world* objects, when most existing image-CAD datasets [9] available for training are limited to images of rendered CAD. Our work continues along this line by fine-tuning a vision-language foundation model specifically for CAD code synthesis.

2.2 Conditional CAD Generation

Conditional methods are those that base their CAD generation on an input – typically text, an image, or both – so that the generated CAD is derived from a user input. For mechanical engineering tasks (e.g., creating CAD from a physical prototype), conditional CAD generation is much more useful than unconditional generation, where CAD is generated by sampling from a learned distribution. Until recently, much existing work on AI for CAD has focused on unconditional generation [7, 8, 23].

All of the works presented in Table 1 explore methods for input-conditional CAD generation. [12, 16, 17, 20] focus on text-conditioned CAD generation. For example, Khan et al. [16] present a direct text-to-parametric CAD framework that accepts text instructions and outputs the corresponding CAD program. While text-conditioned CAD generation allows for user control over the generated CAD, geometries of real-world objects can be difficult to express using natural language. [9, 15] develop models explicitly trained to generate a CAD program given an input image. Combining modalities, [11, 13, 14, 18, 19] all develop methods for image + text conditioned CAD generation. [11] go as far as to study the differences between conditioning VLMs on text versus image + text for CAD generation. While text-only conditioning was surprisingly effective, they found that providing both image and text inputs increasingly paid off for more complex solids. Building on these works and findings, CAD-Coder is conditioned on both input images and text, although, for now, we employ uniform text prompts.

2.3 Editable CAD Representations and Datasets

We also focus on methods and datasets that make use of "editable" CAD representations, parametric CAD programs that encode the design history of the geometric model. These editable representations are more useful to mechanical designers than other representations of 3D solids – such as voxels [2], point clouds [3], or meshes [4] – since parametric values can easily be modified after generation.

Refs. [8, 9, 13–15] train models that output editable CAD in the form of DSL-based CAD programs. These CAD programs treat CAD commands as a language or as a grammar. For example, [16] and [9] use the CAD program representation defined by [8], where a CAD program is represented as a sequence of CAD command-parameter vectors, with individual commands and parameters comprising a vocabulary. Limited to 2D, [13] use a different, but similarly structured, CAD program representation. Ref. [14] also define their own DSL for CAD commands (OpenECAD operations) which they later convert into PythonOCC code for execution. These CAD program representations are challenging because they cannot generate solids without conversion scripts that translate them into code-based formats, and they are often incomplete. Ref. [13]'s CAD program representation is limited to sketches (lines, arcs, and circles) plus constraints while [8]'s CAD program representation is limited to the same sketch commands (defined differently) plus extrusion commands. As of yet, no complete – including splines, revolves, sweeps, lofts, fillet, chamfer, etc. – CAD program representation has been defined. Defining a complete DSL-based CAD program representation would be challenging due to the non-linear interdependence and referential nature of many CAD operations.

An alternative to defining a CAD program representation is to utilize an existing, CAD code scripting package. Written in widely-adopted coding languages (e.g. Python, C++), this CAD code representation aligns well with the knowledge base of foundation models. These scripting packages have the benefit of inherently being editable while also offering a complete representation, as they can be used to define any 3D CAD model. [11, 12, 17–20] leverage pre-trained models to output code that can generate CAD using the Blender API, [2] the CadQuery Python package, [3], or the FreeCAD Python API. [4] The Blender API only works in conjunction with the Blender application/GUI, which is primarily designed for mesh-based animations and graphics. CadQuery and FreeCAD are both wrappers around OpenCascade, [5] a powerful, B-rep-based geometric modeling kernel and both can be run as stand-alone, GUI-less Python scripts for parametric CAD generation, an advantage over the Blender API. Given CadQuery's popularity [11, 17, 18, 20] and extensive documentation, we train CAD-Coder to output CadQuery code. Note that in this paper, all references to "CAD code" refer to an editable CAD representation that makes use of a well-established CAD scripting library (e.g. CadQuery), while "CAD program" refers to a commands-as-vocabulary or author-defined DSL representation, such as those used by [8, 9, 13, 14], discussed previously.

The availability of large scale, editable CAD datasets is somewhat limited due to the difficulty of obtaining human-created designs from commercial 3D programs. One of the largest available CAD datasets, ABC dataset [24], contains 1M 3D models in different data formats such as STEP, STL, domain specific language (FeatureScript), and image. However, the ABC dataset includes duplicates and designs that do not produce valid 3D shapes [25, 26]. Many of the available editable CAD datasets in literature are variants of the ABC dataset. For example, the DeepCAD dataset [8] contains approximately $172k$ CAD programs which are derived from a selected subset – those containing prismatic sketch and extrude operations – of 3D solids in the ABC dataset. [16] build on the DeepCAD dataset by pairing the $172k$ CAD models with 660K textual annotations. The GenCAD dataset [9] also builds on the DeepCAD dataset by coupling each of 168k CAD programs with five grayscale, isometric view images (of varying scales) of the rendered CAD. Our dataset, GenCAD-Code, builds on the GenCAD dataset by converting all

[2]https://docs.blender.org/api/current/
[3]https://CadQuery.readthedocs.io/en/latest/
[4]https://wiki.freecad.org/FreeCAD_API
[5]https://dev.opencascade.org/

of the CAD programs into CadQuery code. Some existing CAD code datasets exist, although they are limited in size. [12] generated a dataset of 8k Blender scripts, while [11] created a custom dataset of 5 classes of mechanical parts (e.g., gears, springs) comprised of 5k examples, each consisting of a sketch, image, text, and CadQuery script. Our dataset, GenCAD-Code—comprised of 163k image-CadQuery pairs—significantly expands the scale of image-CAD code datasets.

3. METHODS

In this section, we describe the generation of our GenCAD-Code dataset and the architecture and training of CAD-Coder. We also characterize the evaluation metrics we use and the baselines we select for comparison with CAD-Coder.

3.1 Dataset Generation

To generate our GenCAD-Code dataset, we converted each CAD program in the GenCAD dataset [9] to a CadQuery script. Each GenCAD CAD program consists of a sequence of CAD commands, and each CAD command is represented by $\mathbf{c}_i = (t_i, \mathbf{p}_i)$, where $\mathbf{c}_i \in \mathbb{R}^{17}$, t_i is the command type ($t_i \in$ {Sketch Line, Sketch Arc, Sketch Circle, Extrude}), and $\mathbf{p_i} \in \mathbb{R}^{16}$ are the parameters associated with each command. We write a script that directly converts each of these CAD commands into its corresponding CadQuery code; additional details can be found in our repository. The GenCAD dataset provides five CAD rendered images per CAD program: one isometric view and four scaled isometric views. This process resulted in our GenCAD-Code dataset of 163,671 CadQuery scripts, each matched with five CAD rendered images. There are 147,289 samples in the training set, 7,355 in the test set, and 9,027 in the validation set.

The distribution of token counts for all 163k generated CadQuery scripts can be seen in Figure 2. The average number of tokens per CadQuery script is 611 tokens, and 99.9% of the CadQuery scripts contain less than 3000 tokens. In general, 3D solid complexity corresponds with number of tokens in its CadQuery script (see Figure 2). The right-skewed dataset distribution illustrates that the dataset is imbalanced when it comes to simple and complex examples. Future work will focus on improving dataset balance to include more "complex" examples.

It is important to note that the direct conversion process we use from the GenCAD CAD programs to CadQuery scripts does not necessarily result in the most concise CadQuery code. For example, the GenCAD CAD programs support only three sketch operations: line, arc, and circle. These CAD programs define rectangle sketches using four sequential line commands, and as such, our generated CadQuery scripts do as well. However, there is a more concise way in CadQuery to generate rectangle sketches, namely the `.rect()` method. Future work will explore methods for post-processing CadQuery scripts to leverage CadQuery's native concise representations (e.g., replacing sequential line commands with more efficient methods like rect()), thereby reducing script complexity.

3.2 CAD-Coder Model Architecture and Training

To train CAD-Coder, we use the same architecture as LLaVA 1.5 [27] and follow a two-stage training process similar to the visual instruction tuning method used to train LLaVA [10, 28]. This method synthesizes a pre-trained LLM with a pre-trained vision encoder. We use Vicuna-13B-v1.5, a 13 billion parameter LLM model, as our pre-trained LLM and CLIP-ViT-L-336px as our vision encoder. CLIP-ViT operates on higher resolution images (336x336 pixels) than other CLIP variants, enabling it to capture finer image details.

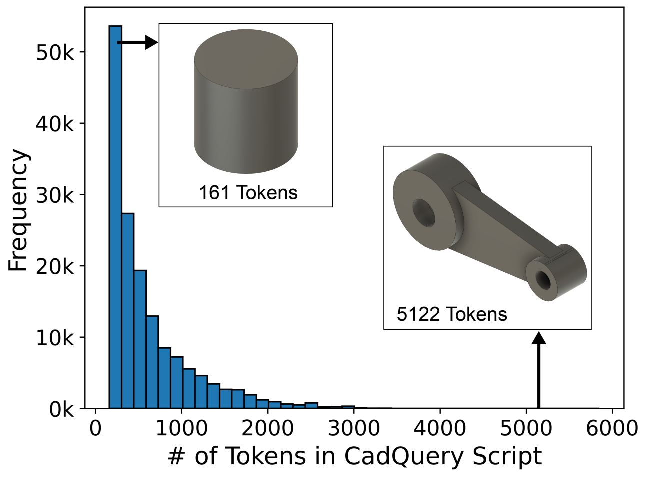


FIGURE 2: Distribution of token counts for the CadQuery scripts in our GenCAD-Code dataset.

Stage 1: Pre-training for Feature Alignment The function of the pre-training phase is to learn a mapping from the image features of the vision encoder to the word embeddings of the LLM. This is done by training a two-layer multi-layer perceptron (MLP) while keeping the vision encoder and LLM weights frozen. The dataset used for this pre-training phase is identical to that used by [10] and consists of 595k samples, each consisting of a question (text-only, $\mathbf{X}_\text{q}$) accompanied by an image ($\mathbf{X}_\text{v}$) and an answer (text-only, $\mathbf{X}_\text{a}$). This dataset was generated by [10] by taking 595K image-caption pairs from the CC3M dataset, where $\mathbf{X}_\text{v}$ is the image, $\mathbf{X}_\text{a}$ is the original caption, and $\mathbf{X}_\text{q}$ is randomly selected from a finite set of questions that asks, in some way, for a description of the image (e.g. $\mathbf{X}_\text{q}$ = "Give a brief description of the image."). The two-layer MLP is trained to maximize the autoregressive likelihood function:

$$p(\mathbf{X}_\text{a} \mid \mathbf{X}_\text{v}, \mathbf{X}_\text{q}) = \prod_{i=1}^{L} p_\theta(x_i \mid \mathbf{X}_\text{v}, \mathbf{X}_{\text{q},<i}, \mathbf{X}_{\text{a},<i}) \tag{1}$$

where L is the tokenized length of $\mathbf{X}_\text{a}$, θ are the trainable parameters (in this case the weights of the two-layer MLP), and $\mathbf{X}_{\text{q},<i}$ and $\mathbf{X}_{\text{a},<i}$ are the question and answer tokens before the current prediction token, x_i.

3.2.1 Stage 2: End-to-End GenCAD-Code Fine-tuning.
After alignment of the vision encoder and word embeddings during pre-training, we conducted supervised fine-tuning. To fine-tune, we use the 147k GenCAD-Code training dataset described in Section 3.1. For each GenCAD-Code

pair, $\mathbf{X}_\text{a}$ is the CadQuery code, $\mathbf{X}_\text{v}$ is the unscaled isometric CAD rendered image, and $\mathbf{X}_\text{q}$ is always constant as: "Generate the CadQuery code needed to create the CAD for the provided image. Just the code, no other words." We leave variation of the text prompt as future work. We filtered out training samples (less than 0.1% of the original training dataset) that would cause the total number of tokens ($\mathbf{X}_\text{q} + \mathbf{X}_\text{v} + \mathbf{X}_\text{a}$) to be greater than 4096, the maximum number of tokens supported by many LLMs. With the vision encoder weights frozen, we again maximize the autoregressive likelihood function (Equation 1), except this time, the trainable parameters, θ, are *both* the two-layer MLP weights as well as all of the LLM's weights.

3.3 Evaluation

We compare CAD-Coder against state-of-the-art image-conditioned, CAD-code-generating baselines using two primary metrics. Due to the computational cost and inference time involved in generating and evaluating CAD programs, we evaluate all baselines on a representative subset of 100 randomly sampled test examples. These samples are from the test set described in Section 3.1 and formatted in the same way as the training data (see Section 3.2.1).

3.3.1 Baselines. We compare our models to both closed-source and open-source VLMs capable of generating CadQuery code. For closed-source VLMs, we evaluate the top-three, best-performing[6] models on the MMMU benchmark leaderboard [29]: GPT-o1 (OpenAI's multimodal reasoning model), GPT-4.5 (OpenAI's advanced multimodal language model), and Gemini-2.0-Pro (Google's latest multimodal model). For open-source VLMs, we evaluate the top-three, best-performing[7] models on the Huggingface OpenVLM Leaderboard [30]: InternVL2_5-78B-MPO (Shanghai AI Lab's large-scale open-source VLM), Ovis2-34B (OpenGVLab's open-source multimodal model), and Qwen2.5-VL-72B (Alibaba's state-of-the-art VLM). To demonstrate the effectiveness of our training data and method for image-conditioned CAD-code generation, we also evaluate LLaVA-v1.5-13B, which is trained identically to our models except stage 2 uses a generic VQA dataset rather than our domain-specialized GenCAD-Code dataset.

All of the baselines were tested using their default inference parameters except for maximum output tokens, which was set to 3450. This number was used so that the total number of tokens (accounting for the fixed-length prompt and image tokens) was 4096, the maximum number possible for many LLMs. An exception was made for GPT-o1 (max_completion_tokens was set to its default value), since its maximum output token quota is used by both the final output and the reasoning process, which happens under-the-hood and a user has no control on it. Since these models were not trained on our GenCAD-Code dataset – which ends each CadQuery script by returning the final solid assigned to the variable "solid" – we appended to the prompt: "Assign the final solid to the variable 'solid' in the last line of code. Do not export or visualize the solid." This addition facilitates automated evaluation of the baselines' generated code. For Qwen2.5-VL-72B, we noticed that it often neglected to include package import statements in its generated scripts, so we also appended to the prompt: "Be sure to include necessary imports." To encourage reproducibility of our results, we evaluate CAD-Coder with a temperature of 0 so that the VLM always selects the most probable next token. For reproducibility, we provide all trained model weights and the exact prompts used during evaluation.

[6]As of March 17, 2025.

[7]As of March 17, 2025.

3.3.2 Metrics. We evaluate all of the baselines and CAD-Coder on two metrics: valid syntax rate (VSR) and the intersection-over-union corresponding with the best alignment between the model-generated solid and the ground-truth solid (IOU_best).

Valid Syntax Rate (VSR): This metric is defined as the percentage of model-generated CadQuery scripts from the test subset that are syntactically valid and return no errors when run as Python files.

IOU_best: To compare the shapes of solid geometries, denoted as Ω here, we first normalize the translation and scale of the models using a normalization operator:

$$n(\Omega) = \{\frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\frac{\text{tr}(\mathbf{I})}{2\times\text{Vol}(\Omega_2)}}} \mid \mathbf{x} \in \Omega\}, \tag{2}$$

where $\mathbf{I}$ is the matrix of inertia and $\mathbf{x}$ is the centroid for the solid Ω. This normalization scales the object by the root mean squared radius of gyration, a choice justified and proven to be optimal in Appendix A and Lemma 2. Note that in this paper, the goal is to generate correct geometry with respect to images. As such, scale information is not inherently an input to the models, and therefore, one must normalize scales for comparison. After this normalization, we align the principle axes of the generated and ground truth solids. The direction of principal axes is ambiguous, and as such we perform all possible valid alignments and use the one with the best Intersection Over Union (IOU) value. The choice of these normalization parameters and the alignment proposed is justified in Appendix A. Unlike commonly used metrics such as Chamfer distance with bounding box or corner alignment (e.g., [8, 9]), where shapes are converted to point clouds and normalized by translating the minimum to the origin and scaling the maximum to 1, our approach provides a higher fidelity and mathematically sound approach for comparing solid objects. While IOU-based measures have been used before to compare solid geometry [11], they have often been overlooked in prior work on datasets such as DeepCAD [31], which relied on the aforementioned Chamfer distance. As demonstrated in Appendix A, we argue that our approach is a more robust and principled alternative. Note that IOU_best is computed over the set of CAD code scripts that execute successfully, excluding those with syntax errors.

4. RESULTS

Below we evaluate the performance of our proposed vision-language model (CAD-Coder) against baselines and additional variants, assessing both code validity (VSR) and geometric accuracy (IOU_best) on the GenCAD-Code test set.

TABLE 2: Evaluation of state-of-the-art, image-conditioned, code-generating models on 100 samples from the GenCAD-Code test set. All models are evaluated on their rate of generating syntactically-valid CadQuery scripts (VSR) and on the accuracy of the resulting 3D solids (IOU_{best}).

Model	VSR ↑	IOU_{best} ↑
Open Source Models		
InternVL2_5-78B-MPO	88%	0.379
Ovis2-34B	83%	0.408
Qwen2.5-VL-72B	94%	0.352
LLaVA-v1.5-13B	0%	0.0
Closed Source Models		
GPT-o1	88%	0.494
GPT-4.5	84%	0.524
Gemini-2.0-Pro	82%	0.445
Ours		
CAD-Coder	**100%**	**0.675**

4.1 Training Details

CAD-Coder is trained on 4 H100 GPUs. We follow the training parameters used for LLaVA 1.5 [28] with the exception of "max_model_length" which we set to 4096 for both stages (rather than 2048) to accommodate longer CadQuery scripts. Stage 1 training took 4.5 hours, and we trained for 1 epoch using a learning rate of 1e-3 and an effective batch size of 256. Stage 2 training took 5.7 hours, and we again trained for 1 epoch using a learning rate of 2e-5 and an effective batch size of 128.

4.2 Comparison with Baselines

We present the results of the evaluation of CAD-Coder against existing image-conditioned, code-generating baselines in Table 2. CAD-Coder outperforms all of the evaluated baselines in terms of both the valid syntax rate and the IOU metric. For each of the 100 samples in the test subset, CAD-Coder generates valid CadQuery code that throws no errors when run as Python scripts. The next-best performer in terms of VSR is Qwen2.5-VL-72B, which has a valid script generation rate of 94%, although this is accompanied by an IOU_{best} approximately half that of CAD-Coder.

In terms of IOU_{best}, CAD-Coder – with a score of 0.675 – performs ~60% better than the next-best open-source model evaluated, Qwen2.5-VL-72B. CAD-Coder also outperforms all of the closed-source models evaluated, and the next-best performing model evaluated (GPT-4.5) has an IOU_{best} score 0.151 lower. To help contextualize these IOU scores, we present various model-generated 3D solids and their corresponding IOU_{best} scores in Figure 3. All of the models are shown in the alignment that maximizes their IOU_{best} score. The disk models in the top row are organized from left-to-right by descending IOU_{best} score. CAD-Coder's generated solid – with a near perfect score of 0.963 – very closely matches the ground truth solid with the exception that the central hole is slightly larger. Even this small difference of 0.04 from a perfect IOU_{best} score (1.0) is visually noticeable, indicating that CAD-Coder's 0.151 improvement in IOU_{best} score over the next-best-performing model is significant. The second row of Figure 3 shows a more complex solid. While CAD-Coder's generated solid is missing the rectangular feature in front, its double triangle shape better matches the ground truth solid than any of the baseline generated solids. This more complex example illustrates how many of the existing image-conditioned, code-generating baselines struggle as CAD model complexity increases. While CAD-Coder achieves a higher IOU_{best} than the evaluated baselines, it still has significant room for improvement, particularly in generating CAD with more complex features.

It is also interesting to note that LLaVA-v1.5-13B – which has the same architecture as CAD-Coder but is trained during stage 2 using a general-purpose VQA dataset rather than the GenCAD-Code dataset – has a score of zero on both VSR and IOU_{best} metrics. While the model generates code-type text reminiscent of CadQuery, none of it is syntactically correct. This result stems from the fact that LLaVA-v1.5-13B—and its underlying LLM, Vicuna-13B-v1.5—seems to have no working knowledge of CadQuery. This demonstrates the usefulness of the architecture (LLaVA v1.5) and training strategy (domain-specific-only for stage 2) that we use for CAD-Coder. Our method produces a domain-specific state-of-the-art model, *even when the pre-trained LLM used has no pre-existing knowledge of the domain-specific task.*

4.3 Generalization Experiments

We hypothesized that a key benefit of training an image-conditioned, CAD generating model using VLMs fine-tuned on CAD code would be in the generalizability of the model. In this section, we investigate the generalizability of CAD-Coder to images of real-world objects and evaluate CAD-Coder's extendability to CAD operations not explicitly included in the GenCAD-Code dataset.

4.3.1 Performance on Images of Real-World Objects. While CAD-Coder was not explicitly fine-tuned to generate CAD code from images of real objects, we hypothesized that CAD-Coder's use of a pre-trained image encoder (CLIP-ViT-L-336px) would help it generalize to this unseen task. To investigate its generalizability to real images, we 3D printed five objects from the test set of GenCAD-Code dataset. We photographed the objects on a wooden table at approximately isometric angles (Figure 4), as all of the GenCAD-Code dataset rendered CAD images capture isometric views of the CAD solids. The 3D CAD models that CAD-Coder generates given these real-image inputs can be seen in the second row of Figure 4. While not perfect matches, these generated CAD models conditioned on the real images broadly agree with the ground-truth 3D solids. These results indicate that CAD-Coder has potential – where trained-from-scratch models struggle – to generalize to CAD generation from real images, even when not explicitly fine-tuned on this task.

The importance of this rendered-CAD image to real-world image translation cannot be overstated. Developing a sufficiently large dataset of real images coupled with CAD code would be extremely expensive, necessitating the time-consuming and costly

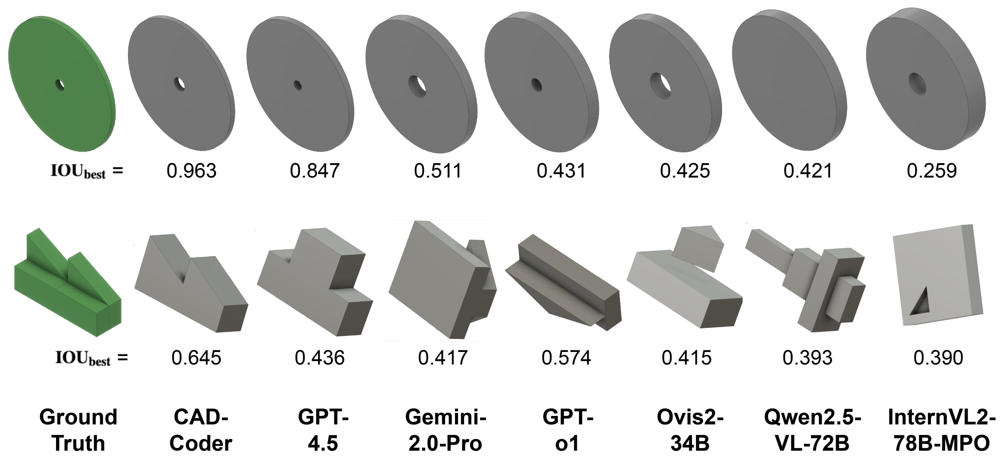


FIGURE 3: Two examples comparing CAD-Coder's generated solids with baseline generated solids. The IOU_{best} score quantifies a solid's similarity to a ground truth solid, where an IOU_{best} of 1 is a perfect score. The solids are depicted in their alignments that yield the IOU_{best} score.

fabrication of hundreds of thousands of physical objects. As such, image-CAD code datasets will likely remain limited to images of rendered CAD models rather than real photographs for the foreseeable future. However, the typical user of an image-conditioned, CAD generating model will rarely input an image of a CAD model, as this implies that they already have the CAD. It is much more likely that a user would want to generate a CAD model given a real photograph. Quantifying the generalizability of CAD generating models to real images is therefore critical.

While CAD-Coder exhibits generalizability to real images, the model still performs noticeably better at generating CAD code given images of rendered CAD models, as seen by the relative score differences between the second and third rows of Figure 4. In some of the real-image-conditioned generation examples (e.g. objects 2 and 4), CAD-Coder predicts the correct CAD operations, but a poor aspect ratio, resulting in a lower IOU_{best} than the rendered-CAD image-conditioned equivalent. We can see this for the second object, where CAD-Coder underestimates the object's side length. CAD-Coder's decreased dimensional accuracy given images of real objects – *colorful* images from slightly *varying perspectives* – is perhaps not surprising given that we train it on *perfectly isometric* and *gray* views. To mitigate this perspective problem, future work will train CAD-Coder on multiple material-rendered-CAD images (from different camera perspectives) per CAD code sample. For objects requiring multiple extrudes (e.g., object 3), CAD-Coder, given the real images, struggles to capture correct CAD operations for these more "complex" solids. Improved generation for these more complicated CAD models might be obtained by incorporating real images of geometric solids into Stage 1 training, so the MLP layers learn a better "description" of these types of images for the LLM.

4.3.2 Performance on Unseen CAD Operations.

We suspected that our approach of fine-tuning a foundation VLM on CAD code would enable the model to extend to CAD operations it did not explicitly see during fine-tuning. To test this theory, we planned an experiment to ask CAD-Coder to add fillets to generated solid models. Filleting is a good test of CAD-Coder's extensibility, as none of the examples in our fine-tuning GenCAD-Code dataset included CadQuery code with fillet commands, but CadQuery supports filleting operations (using the `solid.edges().fillet(fillet_radius)` command). When prompted (in a second query) to add fillets to all edges of a simple rectangular prism CAD model it had previously generated given an input image, CAD-Coder could not do it. Even when the filleting prompt was explicitly provided – reminding the model of the correct syntax of the CadQuery filleting command (see Figure 5) – CAD-Coder would output an extended version of its original code that made no use of the specified fillet command and was syntactically incorrect.

We suspected that CAD-Coder's inability to fillet may stem from the pre-trained LLM's (Vicuna-13B-v1.5) lack of knowledge about CadQuery. To better characterize Vicuna-13B-v1.5's knowledge of CadQuery, we prompted the text-only model to generate CadQuery code of a 4x5x6 unit box with fillets on all edges. The model was not able to generate syntactically correct code. To remedy this issue of CAD-Coder's limited knowledge of CadQuery, we experimented with training CAD-Coder using a different pre-trained LLM. Qwen2.5-Coder-32B-Instruct is the best performing model on an LLM coding leaderboard (Hugginface's bigcode-models-leaderboard [8]), and we tested a smaller variant (14B instead of 32B, due to compute constraints) for evaluation on its ability to generate CadQuery code. When given the same filleted-box text-only prompt that Vicuna-13B-v1.5 struggled with, Qwen2.5-Coder-14B-Instruct output a CadQuery script that produced the desired solid model.

We trained another model, CAD-Coder-Qwen2.5-14B, following the same architecture and training procedure as CAD-Coder except Qwen2.5-Coder-14B-Instruct was used as the pre-

[8]https://huggingface.co/spaces/bigcode/bigcode-models-leaderboard, as of March 17, 2025

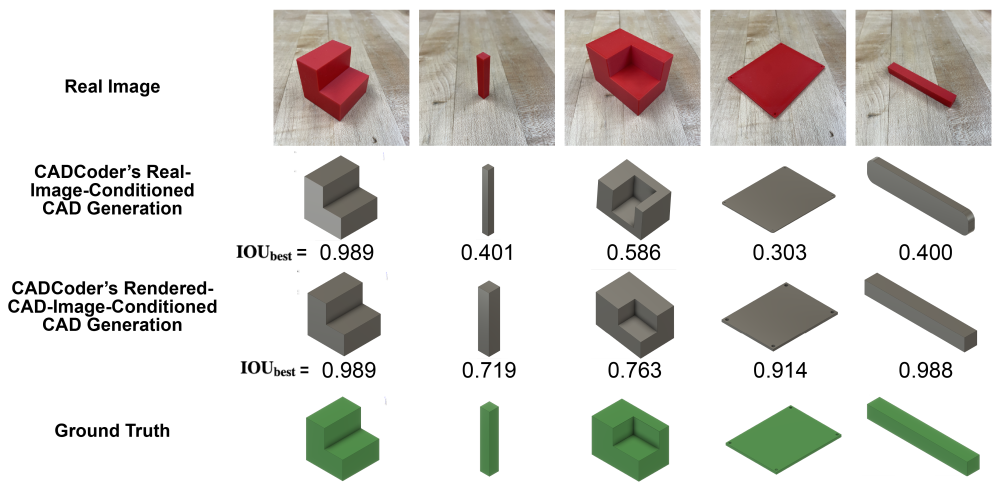

Real Image
CADCoder's Real-Image-Conditioned CAD Generation
IOU_{best} = 0.989
0.401
0.586
0.303
0.400
CADCoder's Rendered-CAD-Image-Conditioned CAD Generation
IOU_{best} = 0.989
0.719
0.763
0.914
0.988
Ground Truth

FIGURE 4: We test CAD-Coder's generalizability to real-image-conditioned CAD generation, a task not included in the fine-tuning dataset. 1st row: we 3D print several objects from GenCAD-Code's test set and photograph them at approximately isometric views. 2nd row: CAD-Coder's real-image-conditioned CAD generation. 3rd row: CAD-Coder's rendered-CAD image-conditioned CAD generation. 4th row: ground truth solids.

trained LLM in favor of Vicuna-13B-v1.5. Stage 2 training for CAD-Coder-Qwen2.5-14B took 3.31 hours on 8 H100 GPUs. We suspected that this model – given its pre-existing knowledge of CadQuery – would perform significantly better than CAD-Coder at adding fillets to a CAD model's edges. However, when given the same two-query filleting question as CAD-Coder, CAD-Coder-Qwen2.5-14B produced syntactically invalid code, incorrectly using the `.edges()` CadQuery method (see Figure 5). This result was surprising, since we knew that Qwen2.5-Coder-14B-Instruct had accurate knowledge of these methods before its fine-tuning in the CAD-Coder pipeline. This result indicates that CAD-Coder-Qwen2.5-14B may have lost some of its pre-trained CadQuery knowledge during fine-tuning.

In an attempt to help preserve Qwen2.5-Coder-14B-Instruct's pre-trained knowledge, we trained one more variant of CAD-Coder, which we call CAD-Coder-Qwen2.5-14B-LowLR, identical to CAD-Coder-Qwen2.5-14B except with a halved learning rate during Stage 2 training (1e-5 instead of 2e-5). This model also took 3.31 hours to train on 8 H100 GPUs. When given the same two-query filleting questions as the other two CAD-Coder variants, CAD-Coder-Qwen2.5-14B-LowLR successfully adds fillets to the generated solid (Figure 5). This result suggests that with tuning of the Stage 2 training hyperparameters, CAD-Coder-type models have the potential to generalize to CAD operations not explicitly included in the fine-tuning dataset. It is important to note that CAD-Coder-Qwen2.5-14B-LowLR's extendability to unseen CAD operations is not robust. The filleting prompt used (Figure 5) is quite prescriptive; more abstract prompts (e.g. "add fillets to all edges") that do not remind the model about the `.fillet()` method usage do not elicit the same correct behavior from CAD-Coder-Qwen2.5-14B-LowLR. Future work will investigate training strategies that more effectively preserve pre-trained LLM knowledge in VLM fine-tuning.

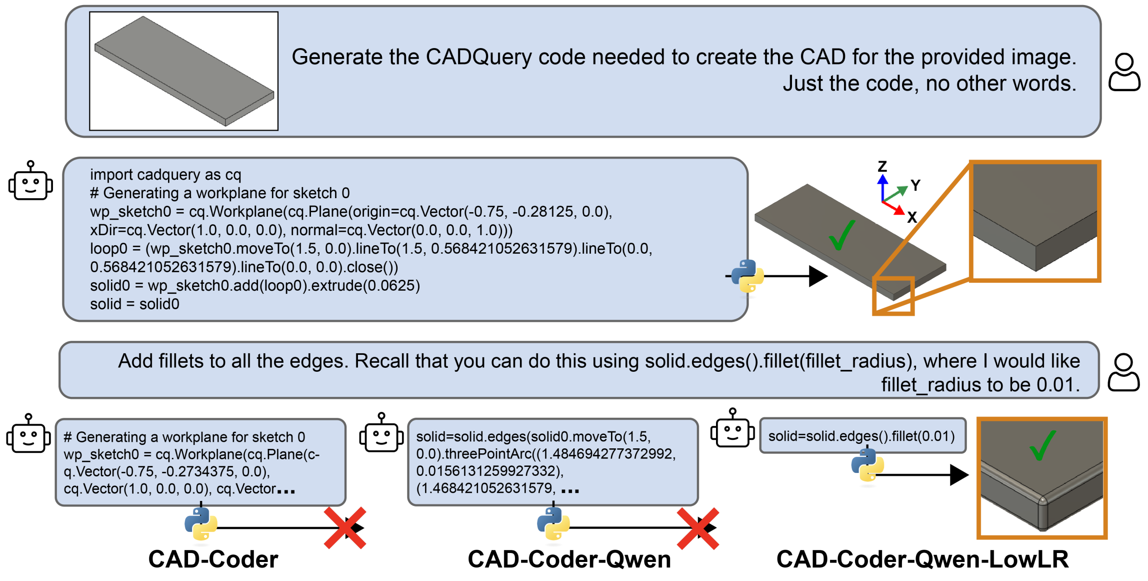
In Table 3, we also quantify CAD-Coder-Qwen2.5-14B and CAD-Coder-Qwen2.5-14B-LowLR according to the VSR and

TABLE 3: Comparison of variants of CAD-Coder with different pre-trained LLMs and learning rates, evaluated by Valid Syntax Rate (VSR) and Intersection-Over-Union (IOU_{best}) scores on the 100-sample test subset.

Model	Pre-trained LLM	Learning Rate	VSR ↑	IOU_{best} ↑
CAD-Coder	Vicuna-13B-v1.5	2e-5	**100%**	**0.675**
CAD-Coder-Qwen2.5-14B	Qwen2.5-Coder-14B-Instruct	2e-5	95%	0.641
CAD-Coder-Qwen2.5-14B-LowLR	Qwen2.5-Coder-14B-Instruct	1e-5	94%	0.592

IOU_{best} metrics used in Section 4.2. Even though it leverages a code-specific pre-trained LLM, CAD-Coder-Qwen2.5-14B unexpectedly performs marginally worse than CAD-Coder on both the VSR and IOU_{best} metrics. The 5/100 syntactically incorrect CadQuery scripts that CAD-Coder-Qwen2.5-14B generates all correspond with relatively complex CAD models, but whose ground-truth scripts fall within the 4096 token limit. In generating these scripts, CAD-Coder-Qwen2.5-14B exceeds the maximum token limit during generation, so the scripts are truncated mid-command. Despite Stage 2 training on the GenCAD-Code dataset (limited to 4096 token examples), CAD-Coder-Qwen2.5-14B's long generation behavior may be a result of the fact that its LLM, Qwen2.5-Coder-14B-Instruct, was pre-trained to handle much longer context lengths (up to 131k tokens). In contrast, CAD-Coder's pre-trained LLM, Vicuna-13B-v1.5, was pre-trained to handle context lengths up to 4096 tokens. While further work is needed to better adapt longer context pre-trained LLMs (>4096 tokens) to our shorter context length dataset, Qwen2.5-Coder-14B-Instruct's longer context limit will be useful in adapting this work to more complex, longer token-length CAD scripts.

In comparison to CAD-Coder-Qwen2.5-14B, CAD-Coder-Qwen2.5-14B-LowLR performs worse in terms of IOU_{best}. This



Generate the CADQuery code needed to create the CAD for the provided image. Just the code, no other words.

```
import cadquery as cq
# Generating a workplane for sketch 0
wp_sketch0 = cq.Workplane(cq.Plane(origin=cq.Vector(-0.75, -0.28125, 0.0),
xDir=cq.Vector(1.0, 0.0, 0.0), normal=cq.Vector(0.0, 0.0, 1.0)))
loop0 = (wp_sketch0.moveTo(1.5, 0.0).lineTo(1.5, 0.568421052631579).lineTo(0.0,
0.568421052631579).lineTo(0.0, 0.0).close())
solid0 = wp_sketch0.add(loop0).extrude(0.0625)
solid = solid0
```

Add fillets to all the edges. Recall that you can do this using solid.edges().fillet(fillet_radius), where I would like fillet_radius to be 0.01.

```
# Generating a workplane for sketch 0
wp_sketch0 = cq.Workplane(cq.Plane(c-
q.Vector(-0.75, -0.2734375, 0.0),
cq.Vector(1.0, 0.0, 0.0), cq.Vector...
```

```
solid=solid.edges(solid0.moveTo(1.5,
0.0).threePointArc((1.484694277372992,
0.0156131259927332),
(1.468421052631579, ...
```

```
solid=solid.edges().fillet(0.01)
```

FIGURE 5: Examples of CAD-Coder variants attempting to add fillets to CAD solids. The figure compares the performance of CAD-Coder, CAD-Coder-Qwen2.5-14B, and CAD-Coder-Qwen2.5-14B-LowLR given identical filleting prompts. Only the LowLR variant correctly applies the fillet operations.

result illustrates that lowering the learning rate – to preserve pre-training knowledge of CadQuery and improve generalization on unseen CAD operations – comes at the cost of reduced performance on the fine-tuning task. The trade-off between improvement on the fine-tuning task and the retention of pre-trained knowledge is a known problem for foundation models [32]. Several fine-tuning strategies [33, 34] have explored alternatives to end-to-end fine-tuning to better balance the two. We intend to explore these in future work.

5. LIMITATIONS, FUTURE WORK AND CONCLUSION

While CAD-Coder demonstrates significant advancements in CAD code generation, it also exhibits some limitations. Firstly, despite its improved generalization to real-world images, the model occasionally struggles to accurately infer precise dimensions and proportions from images taken from varying perspectives. This indicates that further work on perspective and real-world image robustness, possibly through multi-view training or image augmentation strategies, is required. Additionally, although our experiments showed some capability to generalize to unseen CAD operations, such generalization remains heavily prompt-dependent, highlighting the need for enhanced strategies to preserve and leverage pre-existing LLM knowledge during fine-tuning.

Future research will also focus on improving the robustness and generalizability of CAD-Coder by expanding the training dataset and exploring more model architectures. Efforts will be directed towards incorporating multi-view and perspective-invariant image inputs into training to improve model accuracy on real-world images. Developing an extensive test-set of real-world images paired with CAD code would also help to rigorously quantify the performance of CAD-Coder and other models on the real-photo-to-CAD-code task. Other promising directions include exploring continual learning to better retain and extend pre-trained VLM knowledge of CAD operations. Utilizing reasoning-based LLM models or incorporating chain-of-thought logic into the fine-tuning dataset could also improve the accuracy of generated 3D solids.

This paper introduced CAD-Coder, a Vision-Language Model explicitly fine-tuned for generating accurate and editable CadQuery code directly from visual inputs. Leveraging the new GenCAD-Code dataset, we demonstrated substantial improvements over existing image-conditioned, code-generating baselines, achieving state-of-the-art syntactic validity and geometric accuracy. Our findings illustrate the significant potential of fine-tuned VLMs to streamline and enhance engineering design workflows with editable CAD code generation, setting a strong foundation for future research into automated CAD modeling.

ACKNOWLEDGMENTS

The authors gratefully acknowledge MIT-IBM for their partial support of this work. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Alrashedy, Kamel, Tambwekar, Pradyumna, Zaidi, Zulfiqar, Langwasser, Megan, Xu, Wei and Gombolay, Matthew. "Generating CAD Code with Vision-Language Models for 3D Designs." *arXiv preprint arXiv:2410.05340* (2024).

[2] Zhou, Linqi, Du, Yilun and Wu, Jiajun. "3d shape generation and completion through point-voxel diffusion." *Pro-*

ceedings of the IEEE/CVF international conference on computer vision: pp. 5826–5835. 2021.

[3] Luo, Shitong and Hu, Wei. “Diffusion probabilistic models for 3d point cloud generation.” *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*: pp. 2837–2845. 2021.

[4] Jun, Heewoo and Nichol, Alex. “Shap-e: Generating conditional 3d implicit functions.” *arXiv preprint arXiv:2305.02463* (2023).

[5] Liu, Zhengzhe, Wang, Yi, Qi, Xiaojuan and Fu, Chi-Wing. “Towards implicit text-guided 3d shape generation.” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*: pp. 17896–17906. 2022.

[6] Michel, Oscar, Bar-On, Roi, Liu, Richard, Benaim, Sagie and Hanocka, Rana. “Text2mesh: Text-driven neural stylization for meshes.” *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*: pp. 13492–13502. 2022.

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

[8] Wu, Rundi, Xiao, Chang and Zheng, Changxi. “Deepcad: A deep generative network for computer-aided design models.” *Proceedings of the IEEE/CVF International Conference on Computer Vision*: pp. 6772–6782. 2021.

[9] Alam, Md Ferdous and Ahmed, Faez. “Gencad: Image-conditioned computer-aided design generation with transformer-based contrastive representation and diffusion priors.” *arXiv preprint arXiv:2409.16294* (2024).

[10] Liu, Haotian, Li, Chunyuan, Wu, Qingyang and Lee, Yong Jae. “Visual Instruction Tuning.” (2023).

[11] Li, Xingang, Sun, Yuewan and Sha, Zhenghui. “LLM4CAD: Multi-Modal Large Language Models for 3D Computer-Aided Design Generation.” *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 88407: p. V006T06A015. 2024. American Society of Mechanical Engineers.

[12] Du, Yuhao, Chen, Shunian, Zan, Wenbo, Li, Peizhao, Wang, Mingxuan, Song, Dingjie, Li, Bo, Hu, Yan and Wang, Benyou. “BlenderLLM: Training Large Language Models for Computer-Aided Design with Self-improvement.” *arXiv preprint arXiv:2412.14203* (2024).

[13] Wu, Sifan, Khasahmadi, Amir Hosein, Katz, Mor, Jayaraman, Pradeep Kumar, Pu, Yewen, Willis, Karl and Liu, Bang. “Cadvlm: Bridging language and vision in the generation of parametric cad sketches.” *European Conference on Computer Vision*: pp. 368–384. 2024. Springer.

[14] Yuan, Zhe, Shi, Jianqi and Huang, Yanhong. “OpenECAD: An efficient visual language model for editable 3D-CAD design.” *Computers & Graphics* Vol. 124 (2024): p. 104048.

[15] Sharma, Gopal, Goyal, Rishabh, Liu, Difan, Kalogerakis, Evangelos and Maji, Subhransu. “Csgnet: Neural shape parser for constructive solid geometry.” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*: pp. 5515–5523. 2018.

[16] Khan, Mohammad Sadil, Sinha, Sankalp, Uddin, Talha, Stricker, Didier, Ali, Sk Aziz and Afzal, Muhammad Zeshan. “Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts.” *Advances in Neural Information Processing Systems* Vol. 37 (2024): pp. 7552–7579.

[17] Alrashedy, Kamel, Tambwekar, Pradyumna, Zaidi, Zulfiqar Haider, Langwasser, Megan, Xu, Wei and Gombolay, Matthew. “Generating CAD Code with Vision-Language Models for 3D Designs.” *The Thirteenth International Conference on Learning Representations*.

[18] Ocker, Felix, Menzel, Stefan, Sadik, Ahmed and Rios, Thiago. “From Idea to CAD: A Language Model-Driven Multi-Agent System for Collaborative Design.” *arXiv preprint arXiv:2503.04417* (2025).

[19] Mallis, Dimitrios, Karadeniz, Ahmet Serdar, Cavada, Sebastian, Rukhovich, Danila, Foteinopoulou, Niki, Cherenkova, Kseniya, Kacem, Anis and Aouada, Djamila. “CAD-Assistant: Tool-Augmented VLLMs as Generic CAD Task Solvers?” *arXiv preprint arXiv:2412.13810* (2024).

[20] Sun, Yuewan, Li, Xingang and Sha, Zhenghui. “Large Language Models for Computer-Aided Design (LLM4CAD) Fine-Tuned: Dataset and Experiments.” *Journal of Mechanical Design* (2025): pp. 1–19.

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

[22] Radford, Alec, Kim, Jong Wook, Hallacy, Chris, Ramesh, Aditya, Goh, Gabriel, Agarwal, Sandhini, Sastry, Girish, Askell, Amanda, Mishkin, Pamela, Clark, Jack et al. “Learning transferable visual models from natural language supervision.” *International conference on machine learning*: pp. 8748–8763. 2021. PmLR.

[23] Willis, Karl DD, Jayaraman, Pradeep Kumar, Lambourne, Joseph G, Chu, Hang and Pu, Yewen. “Engineering sketch generation for computer-aided design.” *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*: pp. 2105–2114. 2021.

[24] Koch, Sebastian, Matveev, Albert, Jiang, Zhongshi, Williams, Francis, Artemov, Alexey, Burnaev, Evgeny, Alexa, Marc, Zorin, Denis and Panozzo, Daniele. “Abc: A big cad model dataset for geometric deep learning.” *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*: pp. 9601–9611. 2019.

[25] Xu, Xiang, Lambourne, Joseph, Jayaraman, Pradeep, Wang, Zhengqing, Willis, Karl and Furukawa, Yasutaka. “Brepgen: A b-rep generative diffusion model with structured latent geometry.” *ACM Transactions on Graphics (TOG)* Vol. 43 No. 4 (2024): pp. 1–14.

[26] Jayaraman, Pradeep Kumar, Lambourne, Joseph G, Desai, Nishkrit, Willis, Karl DD, Sanghi, Aditya and Morris, Nigel JW. “Solidgen: An autoregressive model for direct b-rep synthesis.” *arXiv preprint arXiv:2203.13944* (2022).

[27] Liu, Haotian, Li, Chunyuan, Wu, Qingyang and Lee, Yong Jae. "Visual instruction tuning." *Advances in neural information processing systems* Vol. 36 (2024).

[28] Liu, Haotian, Li, Chunyuan, Li, Yuheng and Lee, Yong Jae. "Improved baselines with visual instruction tuning." *arXiv preprint arXiv:2310.03744* (2023).

[29] Yue, Xiang, Ni, Yuansheng, Zhang, Kai, Zheng, Tianyu, Liu, Ruoqi, Zhang, Ge, Stevens, Samuel, Jiang, Dongfu, Ren, Weiming, Sun, Yuxuan et al. "Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*: pp. 9556–9567. 2024.

[30] Duan, Haodong, Yang, Junming, Qiao, Yuxuan, Fang, Xinyu, Chen, Lin, Liu, Yuan, Dong, Xiaoyi, Zang, Yuhang, Zhang, Pan, Wang, Jiaqi et al. "Vlmevalkit: An open-source toolkit for evaluating large multi-modality models." *Proceedings of the 32nd ACM International Conference on Multimedia*: pp. 11198–11201. 2024.

[31] Jiang, Shuo, Hu, Jie, Magee, Christopher L and Luo, Jianxi. "Deep learning for technical document classification." *IEEE Transactions on Engineering Management* (2022).

[32] Luo, Yun, Yang, Zhen, Meng, Fandong, Li, Yafu, Zhou, Jie and Zhang, Yue. "An empirical study of catastrophic forgetting in large language models during continual fine-tuning." *arXiv preprint arXiv:2308.08747* (2023).

[33] Mukhoti, Jishnu, Gal, Yarin, Torr, Philip HS and Dokania, Puneet K. "Fine-tuning can cripple your foundation model; preserving features may be the solution." *arXiv preprint arXiv:2308.13320* (2023).

[34] Heyrani Nobari, Amin, Alimohammadi, Kaveh, Arjomand-Bigdeli, Ali, Srivastava, Akash, Ahmed, Faez and Azizan, Navid. "Activation-Informed Merging of Large Language Models." *arXiv e-prints* (2025): pp. arXiv–2502.

[35] Gower, John C and Dijksterhuis, Garmt B. *Procrustes Problems*. Oxford University Press (2004). DOI 10.1093/acprof:oso/9780198510581.001.0001. URL https://doi.org/10.1093/acprof:oso/9780198510581.001.0001.

APPENDIX A. OPTIMAL SOLID MODEL ALIGNMENT

In this section, we detail the continuous solid shape (Rigid Body) alignment problem formally and demonstrate how two shapes are aligned to measure the intersection over union (IOU) for two given shapes. Firstly, we treat solid objects as rigid bodies in 3-dimensional space, that is to say, an arbitrary solid in space is represented as an in-definite set $\Omega \subset \mathbb{R}^3$ with boundary $\Gamma : \partial\Omega$. Given such a solid body we seek to identify the optimal transformation to align any two arbitrary solid geometries.

First, we will analyze the case of two identical (in shape) rigid bodies that have been transformed in space arbitrarily, that is to say, that they have been rotated, scaled, and translated arbitrarily. Under this assumption Lemma 1 implies there exists a relative volume preserving bijection, $f : \Omega_1 \rightarrow \Omega_2$ between the two sets. Note that given the indefinite (continuous) nature of the bodies, hence making the cardinality of sets the same, there exist infinitely many bijections between the two sets. Here we assume that we know the ideal bijection between the identical shapes. Ultimately, the assumption made is that we know the solution to our problem exists, and we formulate the normalization that will allow us to actually find this solution. More intuitively, f maps the points in Ω_1 to the exact same part of the shape in Ω_2 with respect to the shape. This assumption comes with no loss of generality in our shape analysis, and we will not assume anything about f in general other than it's existence and relative volume preservation. The existence of f, is implied by the fact that the shapes are assumed to be identical, meaning they are identical manifolds transformed with an **invertible affine** transformation (See Lemma 1). Ultimately f represents the correspondence between the points in the two sets in 3D Euclidean space, similar to the correspondence often seen in point-based shape matching, made continuous in an abstract sense (no assumptions other than existence are needed for our purposes). Given this bijection, we define the *distance/difference* between two solid bodies as:

$$d(\Omega_1, \Omega_2) := \int_{\Omega_1} \|\mathbf{x} - f(\mathbf{x})\|_2^2 d\mathbf{x} \tag{3}$$

Where $d\mathbf{x}$ is the volume differential in Euclidean space. This is essentially the volume integral of the distance between the points in Ω_1 and Ω_2 given bijection f. Now we formulate our problem of aligning said shapes, namely finding the transformation (inverse of f) such that if applied to Ω_1 maps it onto Ω_2, as:

$$\begin{aligned} \min_{\mathbf{R},s,\mathbf{t}} \quad & d = \int_{\Omega_1} \|s\mathbf{R}\mathbf{x} + \mathbf{t} - f(\mathbf{x})\|_2^2 d\mathbf{x} \\ \text{s.t.} \quad & \mathbf{R} \in \mathbf{SO}(3) \\ & \mathbf{R}\mathbf{R}^\mathbf{T} = \mathbf{I} \\ & \mathbf{t} \in \mathbb{R}^3 \\ & s > 0 \end{aligned} \tag{4}$$

Where $\mathbf{R}$ is a rotation matrix, s is a real number representing the scale, and $\mathbf{t}$ is a translation vector in this setting. In Lemma 2 we attempt to solve this alignment problem for identical shapes. We show that under this assumption of f's existence, we can solve the problem given two solids. However, we do not necessarily have identical shapes when comparing a target CAD model with a candidate reconstruction. However, we can establish a shape normalization scheme that guarantees identical shapes will be in perfect coincidence. Noting Lemma 2, we propose the operator:

$$n(\Omega) = \{\frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\frac{\mathrm{tr}(I)}{2\times\mathrm{Vol}(\Omega_2)}}} \mid \mathbf{x} \in \Omega\}, \tag{5}$$

where $\mathbf{I}$ is the matrix of inertia for solid Ω. Finally, we observed in Lemma 2, that the ideal alignment will align the principle axes of two solids, as such given two solids we first normalize scale and translation using (5) and then align the principle axes of inertial from $\mathbf{I}$ for each solid. Note that the alignment of the principal axes is ambiguous as the direction of the principal axes is not known. As such there are 8 possible ways to align said angles of which 4 are valid rotations in $SO(3)$, meaning that in practice we perform all 4 alignments and pick the one with the best IOU.

Lemma 1. *Let $\Omega_1 \subset \mathbb{R}^3$ be a bounded solid with nonzero volume and let*

$$\Omega_2 = \{\, s\,\mathbf{R}\mathbf{x} + \mathbf{t} \mid \mathbf{x} \in \Omega_1 \,\}$$

be its image under an affine transformation where $\mathbf{R} \in SO(3)$ (so that $\det(\mathbf{R}) = 1$), $s > 0$, and $\mathbf{t} \in \mathbb{R}^3$. Then the mapping

$$f : \Omega_1 \to \Omega_2, \quad f(\mathbf{x}) = s\,\mathbf{R}\mathbf{x} + \mathbf{t},$$

is a bijection satisfying the following relative volume preservation property: For any integrable function $g : \Omega_2 \to \mathbb{R}$,

$$\int_{\Omega_2} g(\mathbf{y})\, d\mathbf{y} = \frac{\mathrm{Vol}(\Omega_2)}{\mathrm{Vol}(\Omega_1)} \int_{\Omega_1} g\big(f(\mathbf{x})\big)\, d\mathbf{x}.$$

Proof. **1. Bijectivity:** The transformation

$$f(\mathbf{x}) = s\,\mathbf{R}\mathbf{x} + \mathbf{t}$$

is an affine map. Since $\mathbf{R}$ is a rotation (and thus invertible) and $s > 0$, the matrix $s\,\mathbf{R}$ is invertible. Consequently, f is one-to-one and onto, with the inverse given by

$$f^{-1}(\mathbf{y}) = \mathbf{R}^T \left(\frac{\mathbf{y} - \mathbf{t}}{s} \right).$$

2. Jacobian Determinant and Volume Scaling: Since f is affine, its differential $Df(\mathbf{x})$ is constant and equal to $s\,\mathbf{R}$ for all $\mathbf{x} \in \Omega_1$. Its Jacobian determinant is

$$\det\big(s\,\mathbf{R}\big) = s^3 \det(\mathbf{R}) = s^3,$$

because $\det(\mathbf{R}) = 1$ given $R \in SO(3)$.

3. Change of Variables and Relative Volume Preservation: By the change-of-variable, for any integrable function g defined on Ω_2,

$$\int_{\Omega_2} g(\mathbf{y})\, d\mathbf{y} = \int_{\Omega_1} g\big(f(\mathbf{x})\big) \left|\det\big(Df(\mathbf{x})\big)\right| d\mathbf{x} = s^3 \int_{\Omega_1} g\big(f(\mathbf{x})\big)\, d\mathbf{x}.$$

In particular, choosing $g(\mathbf{y}) \equiv 1$ yields

$$\mathrm{Vol}(\Omega_2) = s^3\ \mathrm{Vol}(\Omega_1) \Rightarrow s^3 = \frac{\mathrm{Vol}(\Omega_2)}{\mathrm{Vol}(\Omega_1)},$$

where $\mathrm{Vol}(\Omega_1) = \int_{\Omega_1} d\mathbf{x}$ and $\mathrm{Vol}(\Omega_2) = \int_{\Omega_2} d\mathbf{x}$ This demonstrates that the mapping f not only provides a bijective correspondence between points in Ω_1 and Ω_2, but it also preserves the relative volume distribution. □

Lemma 2 (Optimal Rigid-Body Alignment)**.** *Let Ω_1 and Ω_2 be two identical in shape rigid bodies in $\mathbb{R}^3$, assumed to be related by a transformation:*

$$f : \Omega_1 \to \Omega_2, \quad f(\mathbf{x}) = s^*\,\mathbf{R}^*\mathbf{x} + \mathbf{t}^*$$

Define

$$\min_{\mathbf{R}\in SO(3),\, s>0,\, \mathbf{t}\in\mathbb{R}^3} \int_{\Omega_1} \left\| s\,\mathbf{R}\,\mathbf{x} + \mathbf{t} - f(\mathbf{x}) \right\|_2^2 d\mathbf{x}.$$

Then there exists a unique solution:

$$\mathbf{R}^* = \mathbf{V}\mathbf{U}^\mathbf{T},$$
$$s^* = \frac{\sqrt{\frac{\mathrm{tr}(\mathbf{I_2})}{\mathrm{Vol}(\Omega_2)}}}{\sqrt{\frac{\mathrm{tr}(\mathbf{I_1})}{\mathrm{Vol}(\Omega_1)}}},$$
$$t^* = \bar{x}_2 - sR\bar{x}_1,$$

where $\mathbf{I_1}$ and $\mathbf{I_2}$ are the matrices of inertia for solid Ω_1 and Ω_2 respectively and:

$$\bar{x}_1 = \frac{\int_{\Omega_1} \mathbf{x}\, \mathbf{dx}}{\int_{\Omega_1} \mathbf{dx}}, \quad \bar{x}_2 = \frac{\int_{\Omega_2} \mathbf{x}\, \mathbf{dx}}{\int_{\Omega_2} \mathbf{dx}},$$

$$\mathbf{S} = \int_{\Omega_1} \left(f(\mathbf{x}) - \bar{\mathbf{x}}_2\right)\left(\mathbf{x} - \mathbf{x_1}\right)^\mathbf{T} \mathbf{dx} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\mathbf{T},$$

that achieves this minimum.

Proof. First, we note that by Lemma 1, f is a volume preserving bijection. We will use this fact in our derivation of the solution. To find the ideal translation vector we differentiate the objective with respect to the translation and find the root of the gradient:

$$\frac{\partial d}{\partial t} = \frac{\partial}{\partial t} \int_{\Omega_1} \|s\mathbf{R}\mathbf{x} + \mathbf{t} - f(\mathbf{x})\|_2^2 d\mathbf{x} = 2 \int_{\Omega_1} \left(s\,R\,x + t - f(x)\right)\, d\mathbf{x}. \tag{6}$$

Note that we can move the differentiation operation inside the integral given the Leibniz rule. Given this, we can find the root of the gradient to obtain the optimal translation:

$$\begin{aligned}
\int_{\Omega_1} \left(s\,\mathbf{R}\,x + t^* - f(x)\right)\, d\mathbf{x} &= \\
s\,\mathbf{R} \int_{\Omega_1} x\, d\mathbf{x} + \mathbf{t}^* \int_{\Omega_1} d\mathbf{x} - \int_{\Omega_1} f(x)\, d\mathbf{x} &= \\
s\,\mathbf{R}\,\frac{\int_{\Omega_1} x\, d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}} + \mathbf{t}^* - \frac{\int_{\Omega_1} f(x)\, d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}} &= 0 \\
\Rightarrow \mathbf{t}^* = \frac{\int_{\Omega_1} f(x)\, d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}} - s\,\mathbf{R}\,\frac{\int_{\Omega_1} x\, d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}}&.
\end{aligned} \tag{7}$$

Now let $\bar{x}_1$, and $\bar{x}_2$ be the centroids of Ω_1 and Ω_2 respectively:

$$\bar{x}_1 = \frac{\int_{\Omega_1} x\, \mathbf{dx}}{\int_{\Omega_1} d\mathbf{x}}, \quad \bar{x}_2 = \frac{\int_{\Omega_2} x\, d\mathbf{x}}{\int_{\Omega_2} d\mathbf{x}} \tag{8}$$

by lemma 1 we have:

$$\mathbf{t}^* = \bar{\mathbf{x}}_2 - sR\bar{\mathbf{x}}_1 \tag{9}$$

we can see that the sets Ω_1 and Ω_2 can be normalized with respect to translation by removing the centroids, defining $\tilde{\Omega}_1$ and $\tilde{\Omega}_2$ as:

$$\tilde{\Omega}_1 = \{x - \bar{x}_1 \mid x \in \Omega_1\} \tag{10}$$
$$\tilde{\Omega}_2 = \{x - \bar{x}_2 \mid x \in \Omega_2\} \tag{11}$$

With this normalization, we have the variable change:

$$\tilde{\mathbf{x}} = \mathbf{x} - \bar{\mathbf{x}}_1 \quad \text{and} \quad \tilde{f}(\mathbf{x}) = f(\mathbf{x}) - \bar{\mathbf{x}}_2, \tag{12}$$

which allows us to rewrite our objective as:

$$d(s, \mathbf{R}) = \int_{\Omega_1} \|s\mathbf{R}\tilde{\mathbf{x}} - \tilde{f}(\mathbf{x})\|_2^2 d\mathbf{x}. \tag{13}$$

From Lemma 1, we assumed that there exists $f(\mathbf{x}) = s^*\mathbf{R}^*\mathbf{x} + \mathbf{t}^*$ which means $\tilde{f}(\mathbf{x}) = s^*\mathbf{R}^*\mathbf{x}$ given our solution for $\mathbf{t}^*$. Here we can show that s^* can be deduced given the change of variable with translation. Lemma 1 implies:

$$\int_{\Omega_2} \|\mathbf{x} - \bar{\mathbf{x}}_2\|^2 = \frac{\mathrm{Vol}(\Omega_2)}{\mathrm{Vol}(\Omega_2)} \int_{\Omega_1} \|s^*\mathbf{R}^*(\mathbf{x} - \mathbf{x_1})\|^2 \tag{14}$$

Note $\|\mathbf{Rx}\| = \|\mathbf{x}\|$ given $\mathbf{R} \in SO(3)$, therefore:

$$\int_{\Omega_2} \|\mathbf{x} - \bar{\mathbf{x}}_2\|^2 = s^{*2} \frac{\mathrm{Vol}(\Omega_2)}{\mathrm{Vol}(\Omega_1)} \int_{\Omega_1} \|\mathbf{x} - \bar{\mathbf{x}}_1\|^2 \tag{15}$$

This implies:

$$s^* = \frac{\sqrt{\frac{\int_{\Omega_2} \|\mathbf{x} - \bar{\mathbf{x}}_2\|^2 \, d\mathbf{x}}{\mathrm{Vol}(\Omega_2)}}}{\sqrt{\frac{\int_{\Omega_1} \|\mathbf{x} - \bar{\mathbf{x}}_1\|^2 \, d\mathbf{x}}{\mathrm{Vol}(\Omega_1)}}}. \tag{16}$$

Let $\mathbf{I_1}$ and $\mathbf{I_2}$ be the tensors of inertia for Ω_1 and Ω_2, by definition $\int_{\Omega_1} \|\mathbf{x} - \bar{\mathbf{x}}_1\|^2 \, d\mathbf{x} = \frac{1}{2}\,\mathrm{tr}(I_1)$ and $\int_{\Omega_2} \|\mathbf{x} - \bar{\mathbf{x}}_2\|^2 \, d\mathbf{x} = \frac{1}{2}\,\mathrm{tr}(I_2)$, therefore:

$$s^* = \frac{\sqrt{\frac{\mathrm{tr}(I_2)}{\mathrm{Vol}(\Omega_2)}}}{\sqrt{\frac{\mathrm{tr}(I_1)}{\mathrm{Vol}(\Omega_1)}}}. \tag{17}$$

Note that this is under the Euclidean norm objective, and the L1 norm would yield a different scaling (Rotation would not preserve distance under L1 norm in Euclidean space). Now we can introduce the following symbols to simplify expressions:

$$\mathbf{A} = \int_{\Omega_1} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T d\mathbf{x}, \quad \text{Note: } \int_{\Omega_1} \|\tilde{\mathbf{x}}\|^2 d\mathbf{x} = \mathrm{tr}(\mathbf{A}_1) = \frac{1}{2}\,\mathrm{tr}(\mathbf{I}_1) \tag{18}$$

$$\mathbf{S} = \int_{\Omega_1} \tilde{f}(\mathbf{x})\, \tilde{\mathbf{x}}^T \, d\mathbf{x}, \quad \text{Note: } \int_{\Omega_1} \langle \mathbf{R}\,\tilde{\mathbf{x}},\, \tilde{f}(\mathbf{x})\rangle \, d\mathbf{x} = \mathrm{tr}\Big(\mathbf{R}\,\mathbf{S}\Big) \tag{19}$$

Noting that $\int_{\Omega_1} \|\tilde{f}(\mathbf{x})\|^2 d\mathbf{x}$ is a constant and we will refer to it as C below:

$$d(s, \mathbf{R}) = s^2\,\mathrm{tr}(\mathbf{A}) - 2s\,\mathrm{tr}(\mathbf{R}\,\mathbf{S}) + C \tag{20}$$

We can see that rotation only affects the objective only in the term $-2s\,\mathrm{tr}(\mathbf{R}\,\mathbf{S})$ and since $s > 0$ the optimal rotation $\mathbf{R}^*$ is equivalent to the solution to $\max_{\mathbf{R}^* \in SO(3)} \mathrm{tr}(\mathbf{R}\,\mathbf{S})$, which is an orthogonal Procrustes problem which can be solved by singular value decomposition [35]. Let $\mathbf{S} = \mathbf{U\Sigma V}^T$ be the singular value decomposition of $\mathbf{S}$, then $\mathrm{tr}(\mathbf{R}\,\mathbf{S}) = \mathrm{tr}(\mathbf{V^T R U \Sigma})$ which given all of $\mathbf{U}, \mathbf{V}, \mathbf{R}$ are orthogonal will be maximized when $\mathbf{V^T R U} = \mathbf{I}$, hence:

$$\mathbf{R}^* = \mathbf{V}\mathbf{U}^\mathbf{T} \tag{21}$$

Let:

$$\mathbf{A}_1 = \int_{\Omega_1} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T d\mathbf{x} \tag{22}$$

$$\mathbf{A}_2 = \int_{\Omega_1} \tilde{f}(\mathbf{x})\tilde{f}(\mathbf{x})^T d\mathbf{x}, \tag{23}$$

Note that by definition of the matrix of inertia (namely $\mathbf{I} = \int_\Omega \left(\|\mathbf{x} - \bar{\mathbf{x}}\|^2 \mathbf{I}_3 - (\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T \right) d\mathbf{x}$.), we have:

$$\mathbf{A}_i = \frac{1}{2}\mathrm{tr}(\mathbf{I}_i)\,\mathbf{I}_3 - \mathbf{I}_i \quad \forall i \in \{1, 2\} \tag{24}$$

Moreover, given the assumption of f's existence, plugging in the optimal transformation we have:

$$\begin{aligned} \mathbf{A}_2 &= \int_{\Omega_1} \tilde{f}(\mathbf{x})\tilde{f}(\mathbf{x})^T d\mathbf{x} \\ &= \int_{\Omega_1} s^*\mathbf{R}^*\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\mathbf{R}^{*T} s^* d\mathbf{x} \\ &= s^{*2}\mathbf{R}^* A_1 \mathbf{R}^{*T} \end{aligned} \tag{25}$$

Noting the fact that the eigenvectors of $\mathbf{A_i}$ are the same as $\mathbf{I_i}$ from (24) we can see that interestingly this rotation aligns the principal axes of inertia for the two solids without ambiguity. However, in certain settings, one may choose to perform all possible 4 valid alignments of principle axes (8 possibilities with directional ambiguity of eigenvectors of the matrix of inertia, 4 of which will be valid rotations in $SO(3)$) instead of finding the continuous covariance matrix S and performing a singular value decomposition on it. Hence, the unique minimizing transformation parameters $\mathbf{R}^*$, s^*, $\mathbf{t}^*$ are given by the formulas above, completing the proof. □