CAD-Coder: An Open-Source Vision-Language Model for Computer-Aided Design Code Generation

Anna C. Doris
Massachusetts Institute of Technology,
77 Massachusetts Avenue,
Cambridge, MA 02139, USA

Md Ferdous Alam
Massachusetts Institute of Technology,
77 Massachusetts Avenue,
Cambridge, MA 02139, USA

Amin Heyrani Nobari
Massachusetts Institute of Technology,
77 Massachusetts Avenue,
Cambridge, MA 02139, USA

Faez Ahmed
Massachusetts Institute of Technology,
77 Massachusetts Avenue,
Cambridge, MA 02139, USA

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 approaches are often constrained by limited CAD representations, inability to generalize to real-world images, or low output accuracy due to a lack of domain-specific knowledge. To address these challenges, 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 large-scale dataset that we constructed – GenCAD-Code, consisting of over 163k CAD-model image and code pairs – CAD-Coder outperforms VLM baselines such as GPT-4.5 and Qwen2.5-VL-72B on image-conditioned CAD code generation, achieving a 100% valid syntax rate and the highest accuracy in 3D solid similarity. Notably, our VLM exhibits initial signs of generalizability, generating CAD code from real-world images and utilizing a CAD operation not explicitly seen during fine-tuning. The performance and adaptability of CAD-Coder highlight 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.

Keywords: Computer-Aided Design, Design Automation, Vision-Language Models

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. As a result, CAD modeling represents a significant cost in both time and human expertise. A study by Roy et al. quantified the time engineers spent on design planning (qualitative time) and CAD modeling (quantitative time) when designing 15 aluminum parts for aerospace applications [2]. On average, each part required 116 hours of qualitative time and 80 hours of quantitative time, amounting to nearly five 40-hour work-weeks per part. Notably, the engineers in this study already had 2-4 years of experience using the CATIA CAD software; for less experienced designers, the total time would likely be even greater.

This substantial time and experience barrier has motivated researchers to seek ways to partially automate CAD model creation using learning-based approaches [3–6]. An AI-for-CAD assistant would be particularly useful for engineering designers if it supports *both* image and text-based input. Image-based input, in addition to text, is critical because engineering design is inherently visual and often grounded in physical artifacts. Images of legacy components or third-party supplier parts could enable the reconstruction of lost or unavailable CAD models [7,8]. Photos of physical prototypes could be used to generate initial CAD models that are subsequently refined by designers. Reference photos of existing objects that inspire design thinking could serve as visual prompts for CAD generation. An AI-for-CAD assistant should also output CAD models in an *editable* format. Existing CAD software provides engineers with a parametric, editable feature tree that encodes design intent and enables modification of the design. Preserving this editability is critical, as product development is inherently iterative [9]. CAD representations that support efficient revision and reuse therefore play a role in reducing the cost of design iterations.

Early approaches to AI-driven CAD generation have made progress but face notable limitations. A number of works in recent years have trained ML models from scratch to generate 3D representations, with many focusing on non-editable 3D solid geometry such as meshes or voxels [10–14] and fewer focusing on the generation of editable CAD feature trees [15–17]. However, these trained-from-scratch CAD-specific models [15–17] often struggle to generalize. Without the broad visual and contextual knowledge of modern foundation models, they may fail when inputs deviate from their training distribution (e.g., photographs of real-world objects instead of in-distribution synthetic CAD renderings). Moreover, these models typically output CAD programs defined over narrow domain-specific languages (DSLs), which support only a limited set of operations (sketch and extrude). This representation choice constrains downstream CAD editability, while training from scratch on limited DSLs prevents model generalization to CAD operations not observed during training.

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 GPT-4o [1] (closed-source) or LLaVA [18] (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

Version 1.18, August 7, 2026

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

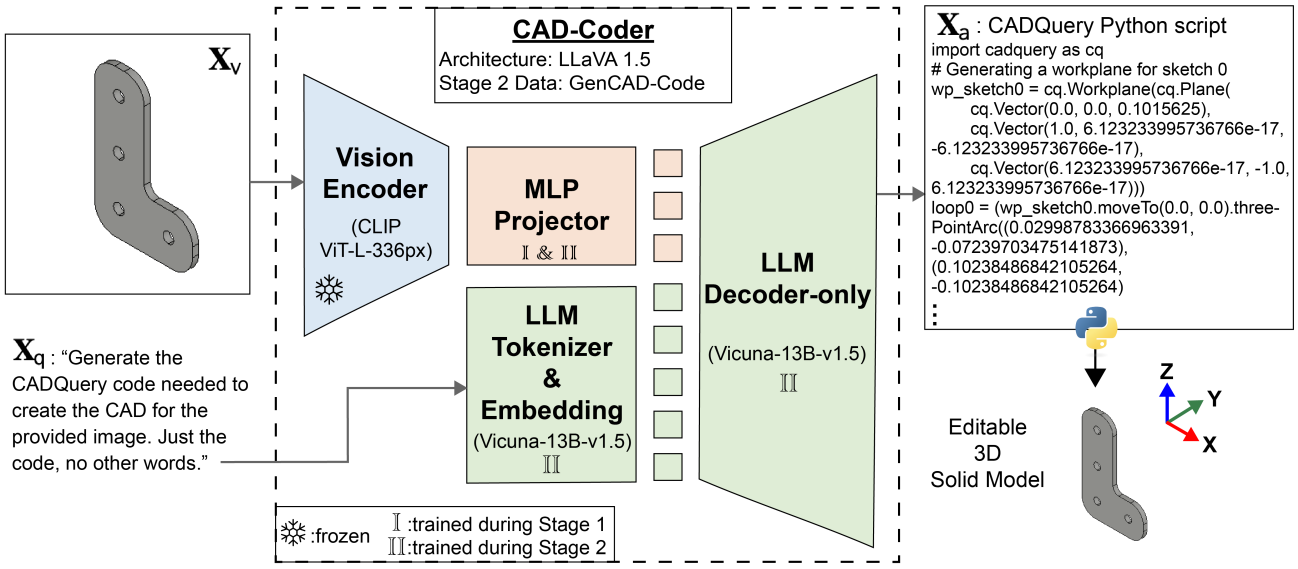

Fig. 1 Overview of CAD-Coder. The VLM accepts an image as input and outputs CadQuery code, which can be run as a Python script to produce an editable, 3D solid CAD model. CAD-Coder has a LLaVA 1.5-type architecture and is fine-tuned on the GenCAD-Code dataset.

have the capacity to output a complete, editable, and operation-rich CAD representation: code for CAD, using an existing 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; however, without domain-specific training, their performance and reliability remains limited (e.g., they may hallucinate incorrect geometry or produce code with syntax errors [19]). Fine-tuning has been used to improve LLMs for text-conditioned CAD code generation [6,20] and for image-conditioned generation of DSL-based CAD programs [21,22]. These approaches, however, are limited by the absence of visual inputs in the former case and by the use of a DSL-based CAD representation in the latter. DSL-based CAD representations can restrict model generalizability, limit downstream CAD editability, and reduce the extent to which pretrained knowledge from large language models can be effectively leveraged. Few works, if any, have fine-tuned a VLM specifically for the direct synthesis of CAD code using a widely-adopted scripting library. This raises the question: to what extent can domain-specific fine-tuning enable an open-source VLM to generate accurate, parametric CAD code from images and to generalize beyond its fine-tuning distribution?

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-the-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) **An Open-Source VLM Fine-Tuned for CAD Code Generation:** We introduce CAD-Coder, a *fine-tuned*, *open-source vision-language foundation model* designed for the generation of *CAD code*. A LLaVA-derived VLM, our model, given image inputs, generates readily-executable Python CadQuery code. Our approach is different from existing methods in that it is a fine-tuned, image-conditioned model that outputs CAD in a code representation (i.e. CadQuery), enabling improved accuracy with respect to existing image-to-CAD-code baselines and demonstrating generalization potential in settings where prior approaches are known to struggle.

(2) **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.

(3) **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 [17] dataset, comprising the largest publicly available dataset, named GenCAD-Code, of CAD code paired with CAD images.

Through these contributions, our work positions foundation

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 [17]	Image	CAD program	3D	Transformer encoder/decoder + Contrastive image encoder + Diffusion prior	From scratch
CSGNet [23]	Image	CAD program	2D/3D	Encoder (CNN) Decoder (RNN)	From scratch
Text2CAD [24]	Text	CAD program	3D	Transformer	From scratch
CADCodeVerify [25]	Text	CAD Code: CadQuery	3D	GPT-4 Gemini 1.5 Pro CodeLlama	None: Multi-Model
IdeaToCAD [26]	Image + Text	CAD Code: CadQuery	3D	GPT-4o	None: Multi-Agent
CAD-Assistant [27]	Image + Text	CAD Code: FreeCAD	3D	GPT-4o	None: Tool Augmented
LLM4CAD [19]	Image + Text	CAD Code: CadQuery	3D	GPT-4/GPT-4V + Debugger	None: Debugger
BlenderLLM [20]	Text	CAD Code: Blender	3D	Qwen2.5-Coder-7B-Instruct	Fine-tuning + Self-improvement
Text2CAD [6]	Text	CAD Code: CadQuery	3D	GPT-3.5 based	Fine-tuning
Cad-VLM [21]	Image + Text	CAD program	2D	Asymmetric transformer encoder-decoder	Fine-tuning
OpenECAD [22]	Image + Text	CAD program	3D	TinyLLaVA	Fine-tuning
CAD-Coder (Ours)	Image + Text	CAD Code: CadQuery	3D	LLaVA 1.5	Fine-tuning

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 [28], 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 [17,23,24] have trained conditioned, editable CAD generating models from scratch. For example, [17] 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 [16] and SkexGen [15] – 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 [25,26] or integrating the models with external tools [20,27]. Alrashedy et al. [25] 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 [26] 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. [27] integrate a VLM "planner" with a Python-API-driven CAD engine (FreeCAD) to enable zero-shot solving of generic CAD tasks. [20] 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.

Many of the existing works discussed in this section rely on closed-source models (e.g., GPT-4) and/or do not release implementation code. In contrast to open-source models, closed-source models limit reproducibility, restrict engineers' abilities to modify them, and raise practical concerns related to data privacy and long-term accessibility. As a result, the development of AI-for-CAD tools that are built on open-source models and released publicly plays an important role in supporting real-world adoption and reproducible research.

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—[20], [6], [21], and [22]—have investigated foundation models' capacity to be fine-tuned for the CAD generation task. [20] 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, [6] fine-tunes GPT-3.5 on a specialized text-to-CadQuery dataset, significantly improving success rates of code generation. Extending to the visual domain, [21] fine-tune a pre-trained vision transformer on a dataset of sketches to improve its ability to generate 2D CAD. Yuan et al. [22] 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 [29]) 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 [17] 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 [15, 16,30].

All of the works presented in Table 1 explore methods for input-conditional CAD generation. [6,20,24,25] focus on text-conditioned CAD generation. For example, Khan et al. [24] 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. [17,23] develop models explicitly trained to generate a CAD program given an input image. Combining modalities, [19,21,22,26,27] all develop methods for image + text conditioned CAD generation. [19] 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 or feature tree of the geometric model. These editable representations are more useful to mechanical designers than other representations of 3D solids – such as voxels [10], point clouds [11], or meshes [12] – since parametric values can easily be modified after generation.

Refs. [16,17,21–23] 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, [24] and [17] use the CAD program representation defined by [16], 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, [21] use a different, but similarly structured, CAD program representation. Ref. [22] also define their own DSL for CAD commands (OpenECAD operations).

An alternative to defining a DSL-based CAD program representation is to utilize CAD code, which makes use of an existing scripting library. Refs. [6,19,20,25–27] leverage pre-trained models to output code that can generate CAD using the Blender API, [2] the CadQuery Python library, [3], or the FreeCAD Python API. [4] 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 [16,17,21,22], discussed previously.

When comparing CAD program representations with CAD code representations, it becomes clear that CAD code offers greater

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

downstream editability. CAD program representations typically support only a narrow set of CAD operations. For example, the representation used by Ref. [21] is limited to sketches (lines, arcs, and circles) and constraints, while the representation used by Ref. [16] is limited to the same sketch commands (defined differently) plus extrusion commands. Extending such DSL-based CAD program representations to include richer CAD operations–such as splines, revolves, sweeps, lofts, fillets, and chamfers–would be challenging due to the non-linear dependencies and referential structure inherent to many CAD features. In contrast, CAD code representations naturally support a broad range of CAD operations, making it easier for engineers to iteratively modify and refine generated designs. In practice, CAD program outputs must ultimately be translated into executable CAD code for visualization and inspection, whereas CAD code can be executed directly, enabling efficient visualization and rapid iteration. Moreover, CAD code is grounded in widely used programming languages and scripting libraries familiar to engineers and well represented in the pre-training corpora of modern vision–language models, whereas DSL-based CAD program representations lack standardized or widely adopted syntax.

Given the enhanced editability of the CAD code representation, we train CAD-Coder to output CadQuery Python code, a wrapper around OpenCascade, [5] a powerful, B-rep-based geometric modeling kernel. We chose the CadQuery library over other CAD code options because of its popularity [6,19,25,26], extensive documentation, and because it can be run as a stand-alone, GUI-less Python script.

2.4 CAD Code Datasets. 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 [31], 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 [32,33]. Many of the available editable CAD datasets in literature are variants of the ABC dataset. For example, the DeepCAD dataset [16] 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. [24] build on the DeepCAD dataset by pairing the 172k CAD models with 660K textual annotations. The GenCAD dataset [17] 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 and DeepCAD datasets by converting all of the human-derived CAD programs into CadQuery code. Some existing CAD code datasets exist. Du et al. [20] generated a dataset of 8k Blender scripts, while Li et al. [19] 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. Rukhovich et al. [34] generate a large-scale dataset of one million CadQuery scripts using a procedural approach. Our dataset, GenCAD-Code—comprised of 163k image-CadQuery pairs—significantly expands the scale of human-derived image-CadQuery 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 [17] 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.

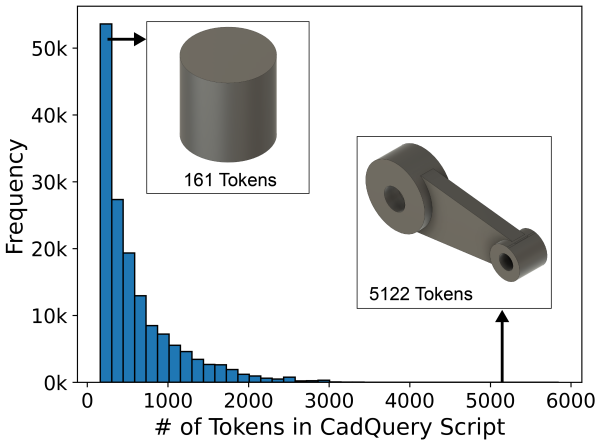

Fig. 2 Distribution of token counts for the CadQuery scripts in our GenCAD-Code dataset.

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 [35] and follow a two-stage training process similar to the visual instruction tuning method used to train LLaVA [18,36]. 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.

[5] https://dev.opencascade.org/

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 [18] and consists of 595k samples, each consisting of a question (text-only, $\mathbf{X}_q$) accompanied by an image ($\mathbf{X}_v$) and an answer (text-only, $\mathbf{X}_a$). This dataset was generated by [18] by taking 595K image-caption pairs from the CC3M dataset, where $\mathbf{X}_v$ is the image, $\mathbf{X}_a$ is the original caption, and $\mathbf{X}_q$ is randomly selected from a finite set of questions that asks, in some way, for a description of the image (e.g. $\mathbf{X}_q$ = "Give a brief description of the image."). The two-layer MLP is trained to maximize the autoregressive likelihood function:

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

where L is the tokenized length of $\mathbf{X}_a$, θ are the trainable parameters (in this case the weights of the two-layer MLP), and $\mathbf{X}_{q,<i}$ and $\mathbf{X}_{a,<i}$ 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, $\mathbf{X}_a$ is the CadQuery code, $\mathbf{X}_v$ is the unscaled isometric CAD rendered image, and $\mathbf{X}_q$ 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 ($\mathbf{X}_q$ + $\mathbf{X}_v$ + $\mathbf{X}_a$) 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 [37]: 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 [38]: 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. These off-the-shelf VLM baselines are evaluated not as optimized task-specific competitors, but to quantify the capabilities of general-purpose VLMs for CAD code generation and the improvements enabled by domain-specific fine-tuning.

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. Baseline models are evaluated using their default or recommended decoding settings. For reproducibility, we provide all trained model weights and the exact prompts used during evaluation.

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 principal 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., [16,17]), 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 [19], they have often been overlooked in prior work on datasets such as DeepCAD [39], which relied on the aforementioned Chamfer distance. As demonstrated in Appendix A, we argue that our approach is a

[6] As of March 17, 2025.
[7] As of March 17, 2025.

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. Regardless of the optimality of alignment and the fact that an image does not inherently include any raw information with respect to the geometry's rotation and scale, thus making this metric mathematically sound, we also report all results for naive IOU– aligning bounding boxes– and chamfer distance in Appendix B and observe that our metric aligns well with both chamfer distance and naive IOU and the overall trends remain the same.

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 [36] 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 an average 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. Median IoU_{best} values reported in Appendix B further demonstrate that CAD-Coder achieves strong typical-case performance across test instances. 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 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,

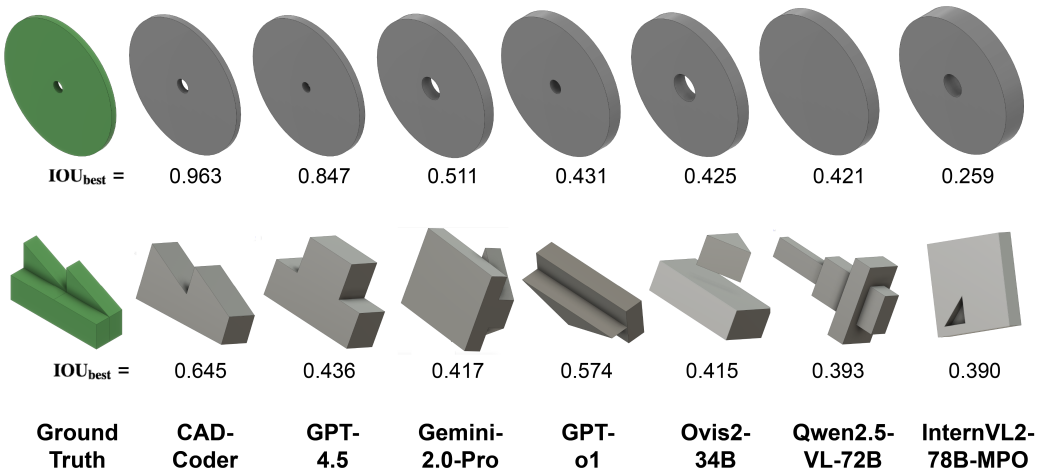

Fig. 3 Two examples comparing CAD-Coder's generated solids with baseline generated solids. The IOU$_{best}$ score quantifies a solid's similarity to a ground truth solid, where an IOU$_{best}$ of 1 is a perfect score. The solids are depicted in their alignments that yield the IOU$_{best}$ score.

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 (Huggingface'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-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 vari-

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

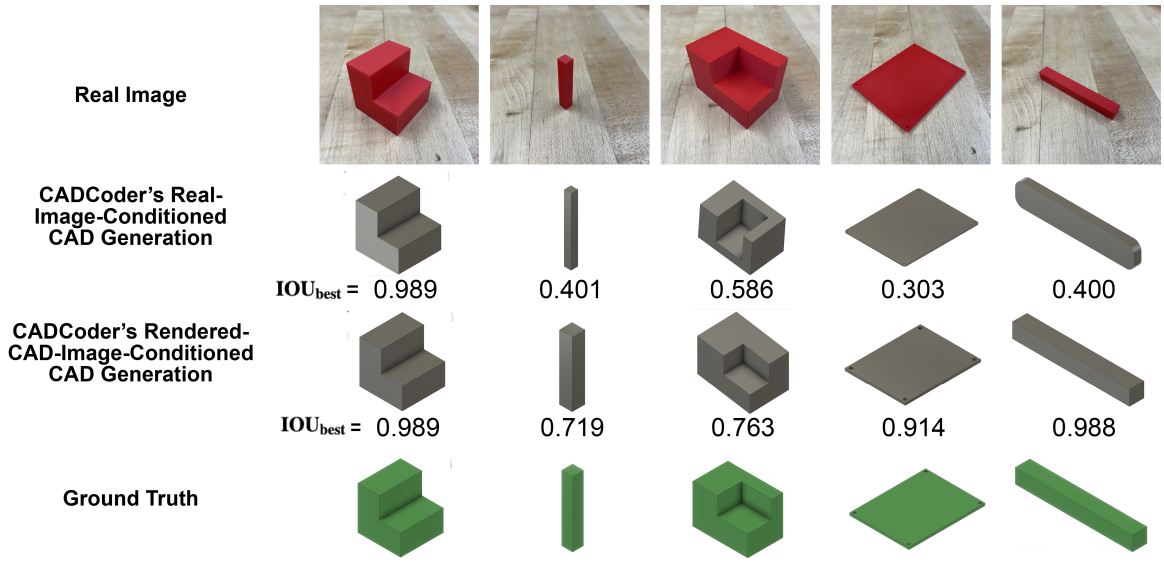

Fig. 4 We test CAD-Coder's generalizability to real-image-conditioned CAD generation, a task not included in the fine-tuning dataset. 1st row: we 3D print several objects from GenCAD-Code's test set and photograph them at approximately isometric views. 2nd row: CAD-Coder's real-image-conditioned CAD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

ants, 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.

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

In Table 3, we also quantify CAD-Coder-Qwen2.5-14B and CAD-Coder-Qwen2.5-14B-LowLR according to the VSR and 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 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 [40]. This phenomenon is known as 'catastrophic forgetting', a central problem in Continual Learning (CL). Beyond the learning-rate adjustment explored in this work, CL literature offers several strategies to mitigate forgetting and preserve the generalist capabilities of foundation models after fine-tuning. These approaches broadly fall into five categories: *regularization*, which constrains parameter updates to retain past knowledge; *replay*, which reuses prior data during training; *optimization*, which modifies the loss function or optimizer; *representation*, which exploits data representations to improve retention; and *architecture*, where models are expanded or augmented as new data arrives [41].

In the context of CAD-Coder, replay is not a viable option, as the original pretraining data is not open-source and its scale is far beyond what is practical. This leaves four promising directions for future work. First, *regularization* via low-rank fine-tuning [42] or selectively freezing model components, which can help preserve general-purpose capabilities while adapting the model for image-to-CAD generation. Second, *representation*-based approaches could leverage the proposed dataset to rephrase the task in a form closer to the model's original training distribution (e.g., by asking an untrained model to rephrase the question and answer), yielding more stable gradients and reduced forgetting. Third, *optimization* strategies include learning-rate control, gradient clipping to limit large parameter updates, or gradient-projection methods using

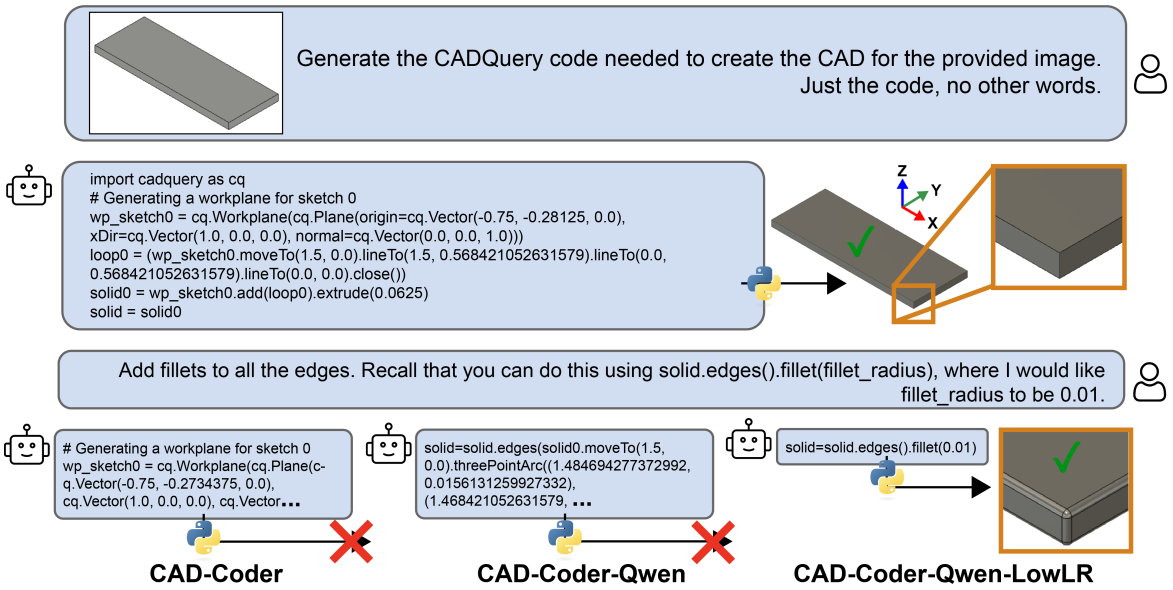

Fig. 5 Examples of CAD-Coder variants attempting to add fillets to CAD solids. The figure compares the performance of CAD-Coder, CAD-Coder-Qwen2.5-14B, and CAD-Coder-Qwen2.5-14B-LowLR given identical filleting prompts. Only the LowLR variant correctly applies the fillet operations.

small calibration sets [43]. Finally, *architectural* approaches could retain the pretrained backbone while injecting lightweight, task-specific modules trained solely for CAD code generation. These directions represent only a subset of possible extensions. Related efforts have already explored forgetting mitigation in foundation models [44], while recent work has proposed model-merging strategies inspired by CL [45], where full fine-tuning is followed by weight merging to retain salient pretrained knowledge. We view these approaches as promising future avenues and intend to investigate them further.

5 Conclusion

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.

Limitations: 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. Finally, as discussed in Section 3.1, the training data is biased toward simple geometries; even the most complex examples in the dataset consist of primarily single-part, sketch-and-extrude models. These limitations reflect the current scarcity and difficulty of obtaining or generating large-scale datasets of complex CAD models in editable, code-based formats.

Future Work: Future research will focus on improving the robustness and generalizability of CAD-Coder by expanding the training dataset and exploring more model architectures. Dataset efforts will be directed toward synthetic and conversion-based approaches for generating more complex CAD models, including parts with richer feature trees and a broader range of CAD operations. The dataset can also be enhanced by 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. The utility of CAD-Coder to engineering designers could be further enhanced by training the model to explicitly support incremental edits and by integrating it with existing CAD software.

Acknowledgment

The authors gratefully acknowledge MIT-IBM Watson AI Lab for their partial support of this work. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship (Grant No. 2141064). 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.

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 indefinite 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, $f : \Omega_1 \to \Omega_2$ 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:

$$d(\Omega_1, \Omega_2) := \int_{\Omega_1} \|\mathbf{x} - f(\mathbf{x})\|_2^2 d\mathbf{x} \tag{A1}$$

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:

$$\begin{aligned} \min_{\mathbf{R},s,\mathbf{t}} \quad & d = \int_{\Omega_1} \|s\mathbf{R}\mathbf{x} + \mathbf{t} - f(\mathbf{x})\|_2^2 d\mathbf{x} \\ \text{s.t.} \quad & \mathbf{R} \in \mathbf{SO}(3) \\ & \mathbf{R}\mathbf{R}^{\mathbf{T}} = \mathbf{I} \\ & \mathbf{t} \in \mathbb{R}^3 \\ & s > 0 \end{aligned} \tag{A2}$$

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:

$$n(\Omega) = \{ \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sqrt{\frac{\mathrm{tr}(I)}{2\times \mathrm{Vol}(\Omega_2)}}} \mid \mathbf{x} \in \Omega \}, \tag{A3}$$

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 (A3) 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

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

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:*

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

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

that achieves this minimum.

Proof. Before continuing to the proof, we must note that the goal of the optimization formulation is to deduce decoupled equations for each of the unknown components of the bijection. Under our assumption that the two rigid bodies Ω_1 and Ω_2 are identical shapes, with an assumed bijection of the form f that relates the two shapes. Under the assumption that f exists, we note that the optimal value for the distance objective minima must be exactly 0. In this proof, we show that without knowledge of the optimal f, given two sets Ω_1 and Ω_2, one can determine the optimal scale, translation, and rotation to align the two solids.

Now, we note that by Lemma 1, f is a volume preserving bijection. We will use this fact in our derivation of the solution. Now, for a given arbitrary s and $\mathbf{R}$, to find the ideal translation vector, we differentiate the objective with respect to the translation and find the root of the gradient:

$$\frac{\partial d}{\partial t} = \frac{\partial}{\partial t}\int_{\Omega_1} \|s\mathbf{R}^*\mathbf{x} + \mathbf{t} - f(\mathbf{x})\|_2^2 d\mathbf{x} = 2\int_{\Omega_1} (s\,\mathbf{R}\,\mathbf{x} + \mathbf{t} - f(\mathbf{x}))\; d\mathbf{x}. \tag{A4}$$

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

$$\begin{aligned}
&\int_{\Omega_1} (s\,\mathbf{R}\,x + \mathbf{t}^* - f(\mathbf{x}))\; d\mathbf{x} = \\
&s\,\mathbf{R}\int_{\Omega_1} \mathbf{x}\,d\mathbf{x} + \mathbf{t}^*\int_{\Omega_1} d\mathbf{x} - \int_{\Omega_1} f(\mathbf{x})\,d\mathbf{x} = \\
&s\,\mathbf{R}\,\frac{\int_{\Omega_1} \mathbf{x}\,d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}} + \mathbf{t}^* - \frac{\int_{\Omega_1} f(\mathbf{x})\,d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}} = 0 \\
&\Rightarrow \mathbf{t}^* = \frac{\int_{\Omega_1} f(\mathbf{x})\,d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}} - s\,\mathbf{R}\,\frac{\int_{\Omega_1} \mathbf{x}\,d\mathbf{x}}{\int_{\Omega_1} d\mathbf{x}}.
\end{aligned} \tag{A5}$$

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

$$\bar{\mathbf{x}}_1 = \frac{\int_{\Omega_1} \mathbf{x}\,\mathbf{dx}}{\int_{\Omega_1} d\mathbf{x}}, \quad \bar{\mathbf{x}}_2 = \frac{\int_{\Omega_2} \mathbf{x}\,d\mathbf{x}}{\int_{\Omega_2} d\mathbf{x}} \tag{A6}$$

by lemma 1 we have:

$$\mathbf{t}^* = \bar{\mathbf{x}}_2 - s\mathbf{R}\bar{\mathbf{x}}_1 \tag{A7}$$

Note that, even without knowledge of s^* and $\mathbf{R}^*$, 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:

$$\tilde{\Omega}_1 = \{\mathbf{x} - \bar{\mathbf{x}}_1 \mid \mathbf{x} \in \Omega_1\} \tag{A8}$$

$$\tilde{\Omega}_2 = \{\mathbf{x} - \bar{\mathbf{x}}_2 \mid \mathbf{x} \in \Omega_2\} \tag{A9}$$

If one were to do this, we can easily see that the optimal translation for alignment of $\tilde{\Omega}_1$ onto $\tilde{\Omega}_2$ will be $\mathbf{t}^* = \mathbf{0}$, thus decoupling it from the rotation and scale. Still, the above formula stands in general. If one were to find the optimal rotation and scale, this equation can be used to determine the optimal translation. Now, with the knowledge of the optimal translation, we continue on to determine the other two components of the transformation.

With the knowledge of the optimal translation for arbitrary scale and rotation, and considering the normalization variable change:

$$\tilde{\mathbf{x}} = \mathbf{x} - \bar{\mathbf{x}_1} \quad \text{and} \quad \tilde{f}(\mathbf{x}) = f(\mathbf{x}) - \bar{\mathbf{x}}_2, \tag{A10}$$

allows us to rewrite our objective decoupled from translation as:

$$d = \int_{\Omega_1} \|s\mathbf{R}\tilde{\mathbf{x}} - \tilde{f}(\mathbf{x})\|_2^2 d\mathbf{x}. \tag{A11}$$

Note that the optimal translation is essentially plugged into this distance by the variable change. Now to differentiating with respect to s we can expand the objective (Note $\langle a, b\rangle$ is equivalent to dot product here since we are operating in Euclidean space):

$$\begin{aligned}
d(s, \mathbf{R}) = s^2 &\int_{\Omega_1} \langle\tilde{\mathbf{x}}, \tilde{\mathbf{x}}\rangle\, dx \\
&- 2s\int_{\Omega_1} \langle\mathbf{R}\tilde{\mathbf{x}}, \tilde{f}(\mathbf{x})\rangle\, dx \\
&+ \int_{\Omega_1} \langle\tilde{f}(\mathbf{x}), \tilde{f}(\mathbf{x})\rangle\, dx.
\end{aligned} \tag{A12}$$

Now differentiating with respect to s gives us:

$$\frac{\partial d}{\partial s} = 2s\int_{\Omega_1} \langle\tilde{\mathbf{x}}, \tilde{\mathbf{x}}\rangle\, d\mathbf{x} - 2\int_{\Omega_1} \langle\mathbf{R}\tilde{\mathbf{x}}, \tilde{f}(\mathbf{x})\rangle\, d\mathbf{x} \tag{A13}$$

Finding the root of the derivative gives us:

$$s^* = \frac{\int_{\Omega_1} \langle\mathbf{R}(\mathbf{x} - \bar{\mathbf{x}}_1), f(\mathbf{x}) - \bar{\mathbf{x}}_2\rangle\, d\mathbf{x}}{\int_{\Omega_1} \langle\mathbf{x} - \bar{\mathbf{x}}_1, \mathbf{x} - \bar{\mathbf{x}}_1\rangle\, d\mathbf{x}}, \tag{A14}$$

which, with some clever algebra, turns into:

$$(s^*)^2 = \frac{\begin{array}{r}\int_{\Omega_1} \langle s^*\mathbf{R}(\mathbf{x} - \bar{\mathbf{x}}_1) - f(\mathbf{x}) + \bar{\mathbf{x}}_2, f(\mathbf{x}) - \bar{\mathbf{x}}_2\rangle\, d\mathbf{x} + \\ \int_{\Omega_1} \|f(\mathbf{x}) - \bar{\mathbf{x}}_2\|^2\, d\mathbf{x}\end{array}}{\int_{\Omega_1} \|\mathbf{x} - \bar{\mathbf{x}}_1\|^2\, d\mathbf{x}}. \tag{A15}$$

Now consider lemma 1, the term $\int_{\Omega_1} \|f(\mathbf{x}) - \bar{\mathbf{x}}_2\|^2\, d\mathbf{x}$ can be integrated in Ω_2 as:

$$(s^*)^2 = \frac{\int_{\Omega_1} \langle (s^*\mathbf{R}(\mathbf{x}-\bar{\mathbf{x}}_1) - f(\mathbf{x}) + \bar{\mathbf{x}}_2, f(\mathbf{x}) - \bar{\mathbf{x}}_2 \rangle \, d\mathbf{x} + \frac{\mathrm{Vol}(\Omega_1)}{\mathrm{Vol}(\Omega_2)} \int_{\Omega_2} \|\mathbf{x}-\bar{\mathbf{x}}_2\|^2 \, d\mathbf{x}}{\int_{\Omega_1} \|\mathbf{x}-\bar{\mathbf{x}}_1\|^2 \, d\mathbf{x}}. \tag{A16}$$

We see that the rotation and optimal scaling are intertwined in this problem, and we cannot integrate in Ω_1 for all terms involving f, thus, we are not able to remove f from the solution of s^*, like we could in determining t^*. However, assuming $\mathbf{R}^*$ in known, given our assumption that f exists, we know that the left hand term in the inner product $s^*\mathbf{R}^*(\mathbf{x}-\bar{\mathbf{x}}_1) - f(\mathbf{x}) + \bar{\mathbf{x}}_2$ will be zero, since $s^*\mathbf{R}^*(\mathbf{x}-\bar{\mathbf{x}}_1) + \bar{\mathbf{x}}_2 = f(\mathbf{x})$. Thus, even though we cannot determine the optimal scale for arbitrary rotations, for the optimal rotation solution to the optimal scale becomes:

$$(s^*)^2 = \frac{\frac{\int_{\Omega_2} \|\mathbf{x}-\bar{\mathbf{x}}_2\|^2 \, d\mathbf{x}}{\mathrm{Vol}(\Omega_2)}}{\frac{\int_{\Omega_1} \|\mathbf{x}-\bar{\mathbf{x}}_1\|^2 \, d\mathbf{x}}{\mathrm{Vol}(\Omega_1)}}. \tag{A17}$$

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:

$$s^* = \frac{\sqrt{\frac{\mathrm{tr}(I_2)}{\mathrm{Vol}(\Omega_2)}}}{\sqrt{\frac{\mathrm{tr}(I_1)}{\mathrm{Vol}(\Omega_1)}}}. \tag{A18}$$

As can be seen, for the optimal rotation, yet to be determined, an optimal scaling emerges, which gives rise to another normalization, which is now with respect to the scale. We see that if each of the sets can be normalized with respect to scale by removing dividing by their root mean squared radius of gyration, defining $\hat{\Omega}_1$ and $\hat{\Omega}_2$ as:

$$\hat{\Omega}_1 = \{ \frac{\mathbf{x}-\bar{\mathbf{x}}_1}{\sqrt{\frac{\mathrm{tr}(I_1)}{\mathrm{Vol}(\Omega_1)}}} \mid \mathbf{x} \in \Omega_1 \} \tag{A19}$$

$$\hat{\Omega}_2 = \{ \frac{\mathbf{x}-\bar{\mathbf{x}}_2}{\sqrt{\frac{\mathrm{tr}(I_2)}{\mathrm{Vol}(\Omega_2)}}} \mid \mathbf{x} \in \Omega_2 \}. \tag{A20}$$

Under such normalization, the optimal scaling we measured above for the transformation of $\hat{\Omega}_1$ onto $\hat{\Omega}_2$ will become unity, thus removing the scaling from each solid. Moreover, with the above normalization, the optimal translation is zero, which also removes that component from our considerations.

Finally, we must determine the optimal rotation. We can introduce the following symbols to simplify expressions:

$$\mathbf{A}_1 = \int_{\Omega_1} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T d\mathbf{x}, \quad \text{Note: } \int_{\Omega_1} \|\tilde{\mathbf{x}}\|^2 d\mathbf{x} = \mathrm{tr}(\mathbf{A}_1) = \frac{1}{2}\mathrm{tr}(\mathbf{I}_1) \tag{A21}$$

$$\mathbf{S} = \int_{\Omega_1} \tilde{f}(\mathbf{x})\,\tilde{\mathbf{x}}^T \, d\mathbf{x}, \quad \text{Note: } \int_{\Omega_1} \langle \mathbf{R}\,\tilde{\mathbf{x}},\ \tilde{f}(\mathbf{x}) \rangle \, d\mathbf{x} = \mathrm{tr}\Big(\mathbf{R}\,\mathbf{S}\Big) \tag{A22}$$

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:

$$d(s, \mathbf{R}) = s^2 \, \mathrm{tr}(\mathbf{A}_1) - 2s \, \mathrm{tr}(\mathbf{R}\,\mathbf{S}) + C \tag{A23}$$

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 [46]. Let $\mathbf{S} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be the singular value decomposition of $\mathbf{S}$, then $\mathrm{tr}(\mathbf{R}\,\mathbf{S}) = \mathrm{tr}(\mathbf{V}^\mathbf{T}\mathbf{R}\mathbf{U}\mathbf{\Sigma})$ which given all of $\mathbf{U}, \mathbf{V}, \mathbf{R}$ are orthogonal will be maximized when $\mathbf{V}^\mathbf{T}\mathbf{R}\mathbf{U} = \mathbf{I}$, hence:

$$\mathbf{R}^* = \mathbf{V}\mathbf{U}^\mathbf{T} \tag{A24}$$

Note that the matrix $\mathbf{S}$ is not known without knowing the optimal transformation f itself. However, we can see how the optimal rotation can be found without knowing this covariance matrix directly.

Now let:

$$\mathbf{A}_1 = \int_{\Omega_1} \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T d\mathbf{x} \tag{A25}$$

$$\mathbf{A}_2 = \int_{\Omega_2} (\mathbf{x}-\bar{\mathbf{x}}_2)(\mathbf{x}-\bar{\mathbf{x}}_2)^T d\mathbf{x} = \frac{\mathrm{Vol}(\Omega_2)}{\mathrm{Vol}(\Omega_1)} \int_{\Omega_1} \tilde{f}(\mathbf{x})\tilde{f}(\mathbf{x})^T d\mathbf{x}, \tag{A26}$$

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:

$$\mathbf{A}_i = \frac{1}{2}\mathrm{tr}(\mathbf{I}_i)\,\mathbf{I} - \mathbf{I}_i \quad \forall i \in \{1, 2\} \tag{A27}$$

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

$$\begin{aligned} \mathbf{A}_2 =& s^{*3} \int_{\Omega_1} \tilde{f}(\mathbf{x})\tilde{f}(\mathbf{x})^T d\mathbf{x} \\ =& s^{*3} \int_{\Omega_1} s^*\mathbf{R}^*\tilde{\mathbf{x}}\tilde{\mathbf{x}}^T\mathbf{R}^{*T} s^* d\mathbf{x} \\ =& s^{*5}\mathbf{R}^* A_1 \mathbf{R}^{*T} \end{aligned} \tag{A28}$$

Noting the fact that the eigenvectors of $\mathbf{A_i}$ are the same as $\mathbf{I_i}$ from (A27), 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 principal 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, and in practice, the rotation can be found by exhaustively aligning principal axes. This completes the proof. □

Appendix B: Additional Experimental Validations

Here, we provide a more comprehensive comparison of results using a variety of metrics commonly used in solid geometry comparison. Specifically, we follow the most common metrics used in many related works, namely chamfer distance (CD) [16] and Intersection Over Union (IOU) on geometries that are only aligned to have the same bounding boxes [19]. Note that the $\mathrm{IOU}_{\mathrm{best}}$ metric that we introduce in this paper takes into account the optimal alignment of geometries. This is a more principled approach for comparing geometries, considering that an image of a part does not include any information regarding scale and orientation, thus aligning the geometries before measuring IOU, as we do in $\mathrm{IOU}_{\mathrm{best}}$,

is more appropriate in such scenarios. Despite this, for consistency with prior works, we also provide the commonly used metrics in Table 4. As is evident, there is solid agreement between IOU_{best} and IOU regardless of the alignment; however, alignment in general leads to slightly better results in most cases, as indeed the aligning geometries optimally is expected to do so. Overall, the main findings in the paper remain unchanged, with CAD-Coder exceeding all other models in both IOU and CD significantly.

We also report median IoU_{best} in Table 4 to characterize the distribution of reconstruction quality across test instances. Median IoU_{best} reflects typical-case performance and is less sensitive to a small number of difficult or failure cases that can disproportionately affect the mean. As shown in Table 4, CAD-Coder exhibits a substantially higher median IoU than mean IoU, indicating consistently high-quality reconstructions for the majority of inputs. In contrast, the other tested models exhibit similar mean and median values, suggesting more uniform—but lower—reconstruction quality across instances.

Table 4 Comparison of CAD-Coder against existing baselines on several metrics beyond IOU_{best} averages: chamfer distance (CD), IOU without optimal alignment (aligned via bounding box), and IOU_{best} medians.

Model	CD ↓	IOU↑	IOU_{best} ↑	IOU_{best} Med. ↑
Open Source Models				
InternVL2_5-78B-MPO	0.0471	0.300	0.379	0.380
Ovis2-34B	0.0482	0.289	0.408	0.400
Qwen2.5-VL-72B	0.0436	0.357	0.352	0.325
LLaVA-v1.5-13B	N/A	0.0	0.0	0.0
Closed Source Models				
GPT-o1	0.0410	0.379	0.494	0.525
GPT-4.5	0.0429	0.431	0.524	0.534
Gemini-2.0-Pro	0.0315	0.411	0.445	0.447
Ours				
CAD-Coder	**0.0197**	**0.663**	**0.675**	**0.792**

References

[1] Alrashedy, K., Tambwekar, P., Zaidi, Z., Langwasser, M., Xu, W., and Gombolay, M., 2024, "Generating cad code with vision-language models for 3d designs," arXiv preprint arXiv:2410.05340.

[2] Roy, R., Kelvesjo, S., Forsberg, S., and Rush, C., 2001, "Quantitative and qualitative cost estimating for engineering design," Journal of engineering design, **12**(2), pp. 147–162.

[3] Ault, H. K., 1999, "Using geometric constraints to capture design intent," Journal for Geometry and Graphics, **3**(1), pp. 39–45.

[4] Ohsuga, S., 1989, "Toward intelligent CAD systems," Computer-aided design, **21**(5), pp. 315–337.

[5] Regenwetter, L., Nobari, A. H., and Ahmed, F., 2022, "Deep generative models in engineering design: A review," Journal of Mechanical Design, **144**(7), p. 071704.

[6] Sun, Y., Li, X., and Sha, Z., 2025, "Large Language Models for Computer-Aided Design Fine Tuned: Dataset and Experiments," Journal of Mechanical Design, **147**(4), p. 041710.

[7] Raja, V. and Fernandes, K. J., 2007, *Reverse engineering: an industrial perspective*, Springer Science & Business Media.

[8] Dori, D. and Tombre, K., 1995, "From engineering drawings to 3D CAD models: are we ready now?" Computer-Aided Design, **27**(4), pp. 243–254.

[9] Ulrich, K. T. and Eppinger, S. D., 2016, *Product design and development*, McGraw-hill.

[10] Zhou, L., Du, Y., and Wu, J., 2021, "3d shape generation and completion through point-voxel diffusion," *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5826–5835.

[11] Luo, S. and Hu, W., 2021, "Diffusion probabilistic models for 3d point cloud generation," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2837–2845.

[12] Jun, H. and Nichol, A., 2023, "Shap-e: Generating conditional 3d implicit functions," arXiv preprint arXiv:2305.02463.

[13] Liu, Z., Wang, Y., Qi, X., and Fu, C.-W., 2022, "Towards implicit text-guided 3d shape generation," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 17896–17906.

[14] Michel, O., Bar-On, R., Liu, R., Benaim, S., and Hanocka, R., 2022, "Text2mesh: Text-driven neural stylization for meshes," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13492–13502.

[15] Xu, X., Willis, K. D., Lambourne, J. G., Cheng, C.-Y., Jayaraman, P. K., and Furukawa, Y., 2022, "Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks," arXiv preprint arXiv:2207.04632.

[16] Wu, R., Xiao, C., and Zheng, C., 2021, "Deepcad: A deep generative network for computer-aided design models," *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6772–6782.

[17] Alam, M. F. and Ahmed, F., 2024, "Gencad: Image-conditioned computer-aided design generation with transformer-based contrastive representation and diffusion priors," arXiv preprint arXiv:2409.16294.

[18] Liu, H., Li, C., Wu, Q., and Lee, Y. J., 2023, "Visual Instruction Tuning," .

[19] Li, X., Sun, Y., and Sha, Z., 2024, "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, American Society of Mechanical Engineers, p. V006T06A015.

[20] Du, Y., Chen, S., Zan, W., Li, P., Wang, M., Song, D., Li, B., Hu, Y., and Wang, B., 2024, "BlenderLLM: Training Large Language Models for Computer-Aided Design with Self-improvement," arXiv preprint arXiv:2412.14203.

[21] Wu, S., Khasahmadi, A. H., Katz, M., Jayaraman, P. K., Pu, Y., Willis, K., and Liu, B., 2024, "Cadvlm: Bridging language and vision in the generation of parametric cad sketches," *European Conference on Computer Vision*, Springer, pp. 368–384.

[22] Yuan, Z., Shi, J., and Huang, Y., 2024, "OpenECAD: An efficient visual language model for editable 3D-CAD design," Computers & Graphics, **124**, p. 104048.

[23] Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S., 2018, "Csgnet: Neural shape parser for constructive solid geometry," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5515–5523.

[24] Khan, M. S., Sinha, S., Uddin, T., Stricker, D., Ali, S. A., and Afzal, M. Z., 2024, "Text2cad: Generating sequential cad designs from beginner-to-expert level text prompts," Advances in Neural Information Processing Systems, **37**, pp. 7552–7579.

[25] Alrashedy, K., Tambwekar, P., Zaidi, Z. H., Langwasser, M., Xu, W., and Gombolay, M., "Generating CAD Code with Vision-Language Models for 3D Designs," *The Thirteenth International Conference on Learning Representations*.

[26] Ocker, F., Menzel, S., Sadik, A., and Rios, T., 2025, "From Idea to CAD: A Language Model-Driven Multi-Agent System for Collaborative Design," arXiv preprint arXiv:2503.04417.

[27] Mallis, D., Karadeniz, A. S., Cavada, S., Rukhovich, D., Foteinopoulou, N., Cherenkova, K., Kacem, A., and Aouada, D., 2024, "CAD-Assistant: Tool-Augmented VLLMs as Generic CAD Task Solvers?" arXiv preprint arXiv:2412.13810.

[28] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., 2017, "Attention is all you need," Advances in neural information processing systems, **30**.

[29] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al., 2021, "Learning transferable visual models from natural language supervision," *International conference on machine learning*, PmLR, pp. 8748–8763.

[30] Willis, K. D., Jayaraman, P. K., Lambourne, J. G., Chu, H., and Pu, Y., 2021, "Engineering sketch generation for computer-aided design," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2105–2114.

[31] Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., and Panozzo, D., 2019, "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.

[32] Xu, X., Lambourne, J., Jayaraman, P., Wang, Z., Willis, K., and Furukawa, Y., 2024, "Brepgen: A b-rep generative diffusion model with structured latent geometry," ACM Transactions on Graphics (TOG), **43**(4), pp. 1–14.

[33] Jayaraman, P. K., Lambourne, J. G., Desai, N., Willis, K. D., Sanghi, A., and Morris, N. J., 2022, "Solidgen: An autoregressive model for direct b-rep synthesis," arXiv preprint arXiv:2203.13944.

[34] Rukhovich, D., Dupont, E., Mallis, D., Cherenkova, K., Kacem, A., and Aouada, D., 2025, "Cad-recode: Reverse engineering cad code from point clouds," *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9801–9811.

[35] Liu, H., Li, C., Wu, Q., and Lee, Y. J., 2024, "Visual instruction tuning," Advances in neural information processing systems, **36**.

[36] Liu, H., Li, C., Li, Y., and Lee, Y. J., 2023, "Improved baselines with visual instruction tuning," arXiv preprint arXiv:2310.03744.

[37] Yue, X., Ni, Y., Zhang, K., Zheng, T., Liu, R., Zhang, G., Stevens, S., Jiang, D., Ren, W., Sun, Y., et al., 2024, "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.

[38] Duan, H., Yang, J., Qiao, Y., Fang, X., Chen, L., Liu, Y., Dong, X., Zang, Y., Zhang, P., Wang, J., et al., 2024, "Vlmevalkit: An open-source toolkit for evaluating large multi-modality models," *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 11198–11201.

[39] Jiang, S., Hu, J., Magee, C. L., and Luo, J., 2022, "Deep learning for technical document classification," IEEE Transactions on Engineering Management.
[40] Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J., and Zhang, Y., 2023, "An empirical study of catastrophic forgetting in large language models during continual fine-tuning," arXiv preprint arXiv:2308.08747.
[41] Wang, L., Zhang, X., Su, H., and Zhu, J., 2024, "A Comprehensive Survey of Continual Learning: Theory, Method and Application," IEEE Transactions on Pattern Analysis and Machine Intelligence, **46**(8), pp. 5362–5383.
[42] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W., 2021, "LoRA: Low-Rank Adaptation of Large Language Models," 2106.09685, https://arxiv.org/abs/2106.09685
[43] Farajtabar, M., Azizan, N., Mott, A., and Li, A., 2020, "Orthogonal Gradient Descent for Continual Learning," *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, S. Chiappa and R. Calandra, eds., Proceedings of Machine Learning Research, Vol. 108, PMLR, pp. 3762–3773, https://proceedings.mlr.press/v108/farajtabar20a.html
[44] Mukhoti, J., Gal, Y., Torr, P. H., and Dokania, P. K., 2023, "Fine-tuning can cripple your foundation model; preserving features may be the solution," arXiv preprint arXiv:2308.13320.
[45] Heyrani Nobari, A., Alimohammadi, K., ArjomandBigdeli, A., Srivastava, A., Ahmed, F., and Azizan, N., 2025, "Activation-Informed Merging of Large Language Models," arXiv e-prints, pp. arXiv–2502.
[46] Gower, J. C. and Dijksterhuis, G. B., 2004, *Procrustes Problems*, Oxford University Press.

List of Figures

List of Tables