CADTalk: An Algorithm and Benchmark for Semantic Commenting of CAD Programs

Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy J Mitra, Changjian Li

CADTalk: An Algorithm and Benchmark for Semantic Commenting of CAD Programs

Haocheng Yuan[1] Jing Xu[1] Hao Pan[2] Adrien Bousseau[3,6] Niloy J. Mitra[4,5] Changjian Li[1]

[1]University of Edinburgh [2]Microsoft Research Asia [3]Inria, Université Côte d'Azur
[4]University College London [5]Adobe Research [6]Delft University of Technology

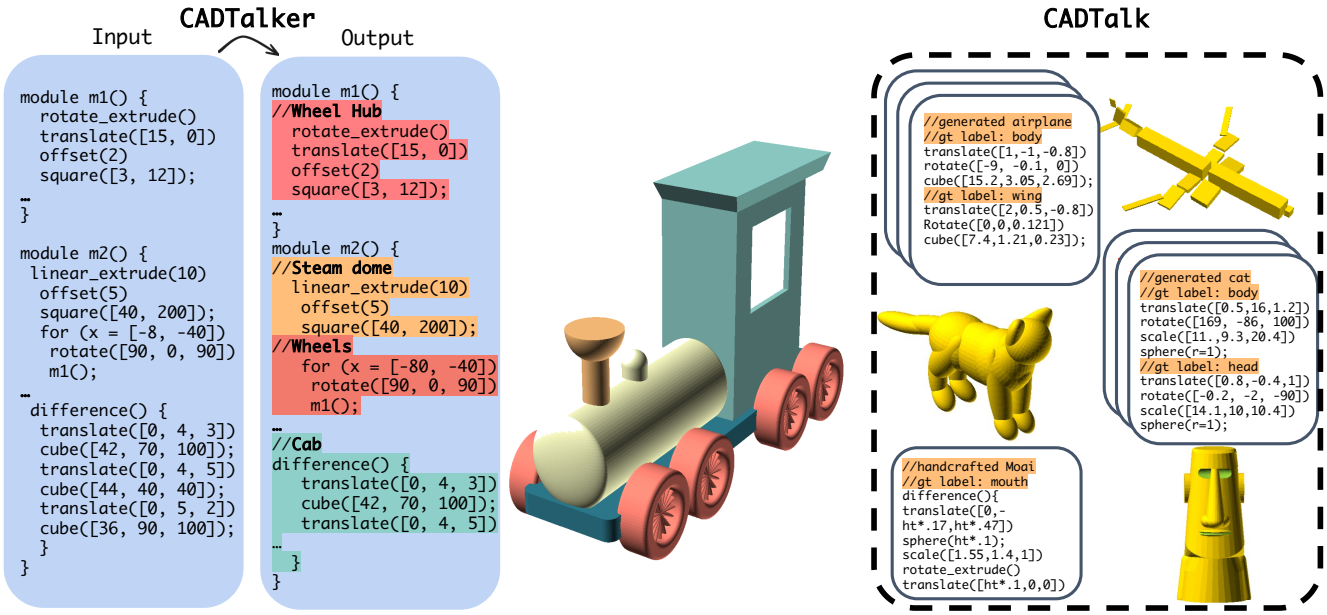


Figure 1. Given a CAD program as input, our algorithm – *CADTalker* – automatically generates comments before each code blocks to describe the shape part that is generated by the block (left). We evaluate our algorithm on a new dataset of commented CAD programs – *CADTalk* – that contains both human-made and machine-made CAD programs (right).

Abstract

CAD programs are a popular way to compactly encode shapes as a sequence of operations that are easy to parametrically modify. However, without sufficient semantic comments and structure, such programs can be challenging to understand, let alone modify. We introduce the problem of semantic commenting CAD programs, wherein the goal is to segment the input program into code blocks corresponding to semantically meaningful shape parts and assign a semantic label to each block. We solve the problem by combining program parsing with visual-semantic analysis afforded by recent advances in foundational language and vision models. Specifically, by executing the input programs, we create shapes, which we use to generate conditional photorealistic images to make use of semantic annotators for such images. We then distill the information across the images and link back to the original programs to semantically comment on them. Additionally, we collected and annotated a benchmark dataset, CADTalk, *consisting of 5,288 machine-made programs and 45 human-made programs with ground truth semantic comments. We extensively evaluated our approach, compared it to a GPT-based baseline, and an open-set shape segmentation baseline, and reported an* 83.24% *accuracy on the new* CADTalk *dataset. Code and data: https://enigma-li.github.io/CADTalk/.*

1. Introduction

Computer-Aided-Design (CAD) is the industry standard for representing 3D shapes as sequences of geometric instructions, also referred to as *CAD programs*. These programs are compact, expressive, and provide a parametric way to

edit shapes. Decades of research have focused on developing domain-specific CAD languages, tools for authoring and robustly executing CAD programs, and even automatically generating them. Popular frameworks include SketchUp, AutoCAD, FreeCAD, and Catia, to name a few.

However, without semantic annotations or comments, CAD programs are challenging to parse and decipher as one has to mentally execute the programs to reveal semantic associations of code blocks with corresponding shape parts. This is particularly problematic when the one using the program differs from the one who authored it, being human or machine. We, as humans, are notorious for leaving sparse comments when writing CAD programs. Machines, on the other hand, can be made to leave systematic comments when generating programs, but these comments may not be semantically relevant nor follow a canonical structure. Without semantic comments and an associated structure, human users find it challenging to interpret and edit CAD programs. Even machines may struggle to learn from programs without a canonical structure and/or semantic association to the underlying shapes.

In this paper, we introduce the problem of *semantic commenting* of CAD programs. Other than the input program and the shape category it represents, we assume access to an execution module to transform any target program to its corresponding 3D shape. Our goal is to segment the program into multi-level code blocks and assign each block a comment indicating the shape part it represents (Fig. 1).

Solving this problem requires overcoming multiple challenges. First, CAD programs, especially the ones designed by humans, contain highly structured constructs (*e.g.*, subroutines, control flows) that organize shape primitives recursively into meaningful parts, and their commenting demands structure parsing in the first place. Second, working directly in the program domain prevents visual recognition of the corresponding shape parts. While executing the program produces a shape that can be rendered, CAD programs lack any material texture and, at best, have pseudo face colors – executing them results in textureless projected images that are challenging to interpret by vision models trained on photographs. Third, CAD programs encode a vast array of shapes across arbitrary categories. Managing this diversity requires an approach that can generalize beyond a pre-defined set of semantic labels.

We introduce *CADTalker* for semantic commenting of CAD programs (Fig. 1), which we achieve by combining program parsing with visual-semantic analysis afforded by recent advances in large language models (LLMs) and foundational vision models. We first handle the nested program structures by performing a syntax tree analysis that identifies commentable code blocks at multiple levels of program constructs. Then, we address the second challenge by leveraging conditional image generation to translate textureless CAD renderings into photorealistic images, which vision models can handle. We employ this photorealistic image synthesizer to render the CAD program from multiple views, resulting in a rich visual depiction of the shape produced by the program. We address the third challenge by using LLMs and vision models to segment and annotate the images with open-vocabulary labels. Finally, we aggregate the segmentation and part labels from the multiview images and transfer them back to relevant code blocks.

To evaluate our method and to foster future research on this problem, we also created *CADTalk* – a new benchmark for semantic commenting of CAD programs. We collected 5300+ programs from a variety of data sources (online repositories [3, 5], and shape abstraction algorithms [28, 41]), comprising human-designed and machine-generated CAD programs. We semi-automatically annotated these programs to provide ground truth comments. Finally, we propose evaluation metrics to score any semantic commenting approach. Based on this dataset, we have conducted statistical evaluation, a comprehensive ablation study inspecting several core components of our method, and comparisons with a GPT-based approach and an open-vocabulary shape segmentation method (i.e., PartSLIP [26]). All the evaluations demonstrate that *CADTalker* sets a good baseline for future research on this problem.

In summary, this paper introduces the new task of semantic commenting CAD programs and the first benchmark and algorithm dedicated to this task.

2. Related Work

CAD programs. CAD programs represent 3D shapes as sequences of geometric instructions, which brings advantages in terms of compactness, editability, and modularity. However, CAD programs are notoriously difficult to author, as the effect of each instruction on the resulting shape can be difficult to foresee. Recent work in interactive modeling eases CAD authoring thanks to differentiable execution [8, 31], which allows editing program parameters via direct shape manipulation, but does not allow modification of the program structure. Closer to our goal is the recent work by Kodnongbua et al. [22], who use large pre-trained image and language models to discover semantically meaningful parameter ranges from multi-view renderings of a CAD program. Using similar ingredients, we target the complementary task of generating semantic comments for code blocks that might include several parametric instructions.

Another stream of research aims at automatically generating CAD programs for reverse engineering [10, 43, 46], sketch-based modeling [23], or generative modeling [17, 44, 45], see [39] for a recent survey. While algorithms exist to structure the raw code generated by such methods, for instance by identifying compact macros that encapsulate repetitive parts [19, 33], in the absence of comments,

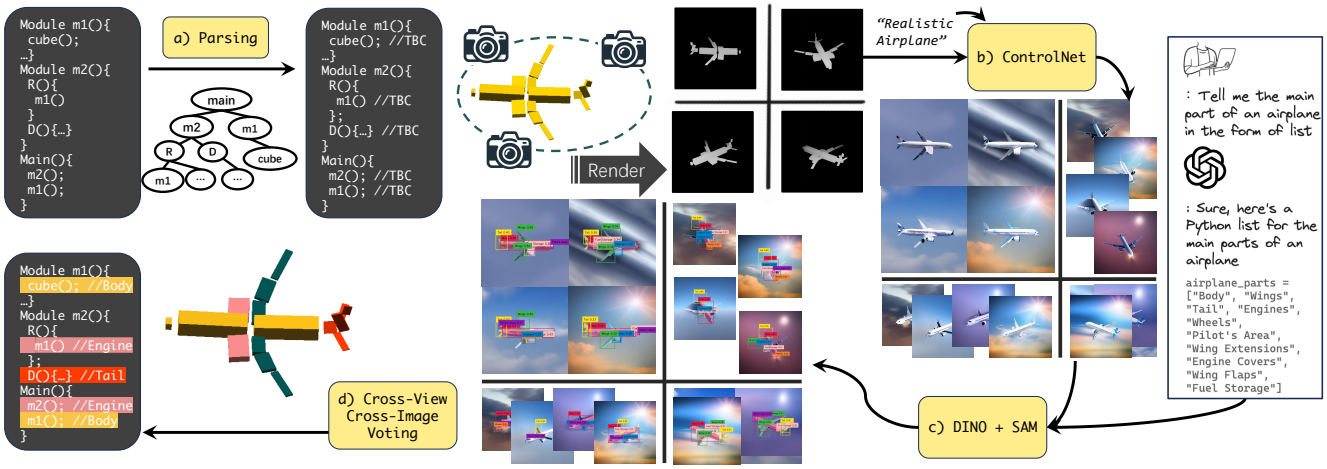


Figure 2. **Algorithm overview.** We first parse the input program to identify commentable code blocks, marked with TBC (a). We then execute the program and render the resulting shape under several viewpoints to obtain multiview depth maps, which we convert into realistic images using image-to-image translation (b). In addition, we obtain a list of part names of the shape from ChatGPT. We use these labels to segment semantic parts in the images using computer vision foundation models (c). Finally, we aggregate this semantic information across views by linking it to the code blocks that correspond to the segmented parts (d).

machine-made code is difficult to interpret and build upon. We designed our benchmark dataset to include both human-made and machine-made CAD programs to foster research.

Program summarization. There is a large body of work on the automatic generation of comments for generic programming languages (C/C++, Python, Java, etc.). While early methods were based on template matching and text retrieval [13], the field then adopted sequence-to-sequence neural models [4, 16] and most recently LLMs [9]. Such models are typically trained on large corpus of commented code, collected from code sharing platforms like StackOverflow [16] and GitHub [9]. Unfortunately, we are not aware of any large dataset of commented CAD programs that could be used to train such language-specific models. Besides, sequence-to-sequence models reason on raw code snippets rather than on their execution, and as such are not equipped to analyze the visual output of CAD programs. In contrast, we leverage CAD execution and rendering to convert the problem of code commenting into the problem of semantic image segmentation, allowing us to build upon pre-trained image models, and making our method agnostic to the input CAD language. Nevertheless, we experimented with few-shot training strategies of an LLM for code commenting [2, 6], and observed a surprising ability of the LLM to capture the geometric structure of an object category from only one example CAD program, although it struggles to generalize to different categories.

Semantic segmentation of images and shapes. We formulate commenting of CAD programs as a semantic segmentation task, making our approach related to prior work on shape segmentation and labeling. Recent approaches to label parts of 3D shapes employ deep learning, using neural networks tailored to voxel grids [42], 3D meshes [11, 14], point clouds [36]. Closest to our approach is the work of Kalogerakis et al. [20, 40], who render the 3D shape from multiple views to leverage 2D CNNs for segmentation tasks. We use pre-trained *open-vocabulary* object detection [27] and segmentation [21]. Open-vocabulary methods rely on large image-language models [37] to recognize arbitrary objects in images [25], avoiding the need for a predefined set of labels. Our approach also relates to the recent PartSLIP [26], which combines multi-view rendering and a pre-trained image-language model GLIP [24] to segment 3D scans. A key difference is that the CAD programs we target are not equipped with realistic colors and textures, preventing the direct application of image models trained on photographs. We tackle this by converting our synthetic renderings into realistic images using image-to-image translation [47]. This strategy allows our approach to outperform a *concurrent* zero-shot mesh segmentation method [1] (see comparison in the supplementary).

3. Commenting Programs with *CADTalker*

Given a CAD program as input, our goal is to automatically insert comments around CAD instructions, such that these comments describe the semantic parts of the shape that the CAD instructions produce upon execution. For example, in Figure 1, the comments specify instructions responsible for the wheels and cab of a train.

Recognizing object parts from CAD instructions is a very difficult task, for both humans and machines, because code instructions only describe shapes indirectly, via geometric functions. Our key idea is to move the problem away from the program domain, towards the image domain where visual inference is much easier. To do so, we leverage CAD

execution to produce the 3D shape corresponding to the program, we perform multiview rendering to obtain representative images of this shape, and we run computer vision models on these images to identify semantic parts and their labels. We then propagate the labels in the opposite direction, from images to the original CAD instructions. Fig. 2 illustrates the main steps of this cross-domain process. We next describe how we leverage off-the-shelf foundation models to achieve these tasks.

While there exist many different CAD domain-specific-languages, and the algorithm we propose is agnostic to the chosen language, we built our prototype implementation around OpenSCAD [30] – a free CAD software based on Constructive Solid Geometry (CSG).

3.1. Realistic Multiview Rendering

At the core of our approach is the idea of leveraging computer vision models trained on photographs to recognize semantic parts in CAD programs. To do so, our first step is to execute the program to produce a 3D shape, and render images of that shape from ten representative viewpoints. We distribute these viewpoints along a ring slightly above the shape, as shown in Fig. 2.

However, many CAD programs only describe the geometry of a shape and do not include realistic textures and materials. Furthermore, some of the CAD programs we consider represent shapes in a simplified manner, missing geometric details or exhibiting spurious gaps between parts. As a result, the images we obtain by directly rendering the program output do not look realistic, and as such are not well recognized by computer vision models (see the ablation study in Sec. 5.2). We tackle this challenge by translating these renderings into photorealistic images using ControlNet [47] – a method that adds image conditioning to a large text-to-image diffusion model. Specifically, we render a depth map of the shape from each of the viewpoints, and we instruct ControlNet to turn each depth map into a realistic image of the shape, providing the category name of the object as a complementary text prompt.

In contrast to related work that seeks to imbue 3D shapes with realistic texture maps [7, 38], our application scenario does not necessitate different views to share a consistent appearance. On the contrary, we observed that the subsequent step of recognizing object parts in the resulting images benefits from diversity in appearance, since object parts that might be ambiguous under a specific appearance might become recognizable under a different appearance. We build on this insight to further increase diversity by running ControlNet with four different seeds on each of the ten views, resulting in 40 realistic images of the shape in total. We also note that for highly abstract shapes, the quality of the image-to-image translation increases when we process the depth maps with a morphologic closing [15] to fill in small gaps (see supplemental materials for parameter settings). Fig. 3 illustrates the realistic images we obtain.

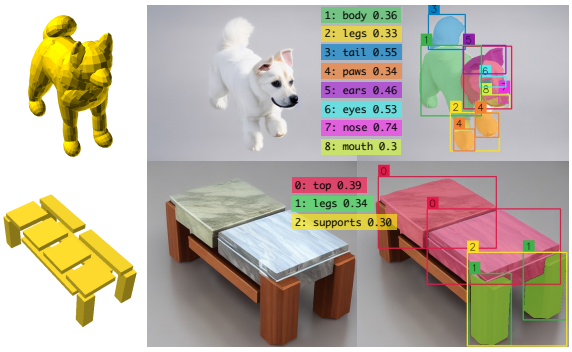


Figure 3. Given shapes (left) after executing programs, we use ControlNet [47] to convert rendered depth maps into realistic images (middle), which form a valid input for detection and segmentation models trained on photographs [21, 27] (right).

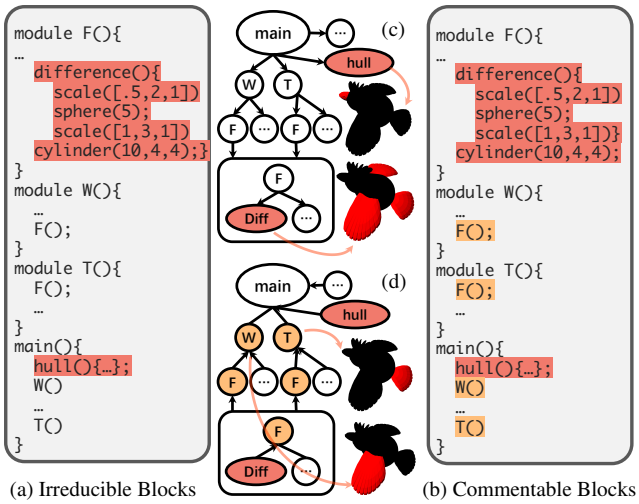
3.2. Part Detection and Segmentation

Given realistic images of the shape, our next step is to segment each image into semantically meaningful parts. We do so by leveraging both language and image foundation models. Since we do not want to restrict ourselves to a fixed list of part labels, we turn to recent open vocabulary detection algorithms to identify relevant parts. Specifically, we use Grounding DINO [27], a method that takes as input an image and the name of an object part of interest, and outputs a bounding box of that part, if present in the image. While users of our system can provide the list of parts to be detected, we found that ChatGPT-v4 [34] provides good suggestions of parts given the name of an object category. We then convert each bounding box into a pixel-wise segment by feeding the image and the bounding box to the Segment Anything Model (SAM) [21]. We denote S_l^i the segment predicted for a given label l on a given image i. Fig. 3 shows the segments and labels predicted for representative images.

3.3. Part Label Voting

The last step of our algorithm consists of aggregating the semantic information of all 40 images of the shape and transferring this information to the corresponding code blocks.

Program Parsing. We first parse the input program to identify all *commentable* code blocks. Specifically, we construct the syntax tree of the input program (Fig. 4c) and traverse the tree from the top downward using breadth-first search until we reach an *irreducible* block, *i.e.,* a sequence of instructions that corresponds either to a single geometric primitive (cuboid, ellipsoid, etc.) or to a difference, intersection, or hull operation on primitives (Fig. 4a). We label the irreducible blocks as commentable leaf nodes and traverse the tree upward to collect all commentable blocks as nodes parent of commentable nodes (Fig. 4b,d).



(a) Irreducible Blocks (b) Commentable Blocks

Figure 4. **Program parsing.** Irreducible blocks are basic-level geometric primitives and their direct compositions (a), while commentable blocks are code blocks of different compositional levels that correspond to semantic comments (b). The downward traversal of the syntax tree is used to identify irreducible blocks (c) and the upward traversal to collect commentable blocks (d). Exemplar masks of commentable blocks are shown in (c) and (d) in red.

Mapping code blocks to image pixels. For each block to be commented on, we render 10 views of the shape using a white color for the block of interest and a background color for all other blocks (see Fig. 4 (c)(d), where we rendered the blocks in red for visualization purpose). This procedure results in a set of binary masks $\{M_b^v\}$ that indicate for each view v the visible pixels of block b, providing us with an explicit mapping between image pixels and code blocks. Fig. 2 illustrates the view setup, detailed view angles can be found in the supplemental material.

Aggregating semantic labels. We represent all possible label assignments via a matrix C, where each entry $C(b,l)$ quantifies the confidence of block b to be assigned label l. We fill in this matrix by accumulating labeling confidence over all 40 images of the shape, in three steps. In a first step we compute, for each image i generated from view v, the confidence of label l to be assigned to block b as:

$$C^i(b,l) = C_{DINO}(i,l) \times IoU(M_b^v, S_l^i), \qquad (1)$$

where $C_{DINO}(i,l)$ is the confidence of label l in image i provided by Grounding DINO, S_l^i is the segmentation mask provided by SAM, M_b^v is the binary mask rendered for block b in view v, and IoU is the Intersection-over-Union. In a second step, we sum the confidence scores obtained for all 4 images of each view. Finally, in a third step, we sum the confidence scores over all 10 views. We perform this aggregation in three steps to introduce intermediate filtering of poor labels. In practice, after each step, we set to zero the confidence of any label having a confidence below a threshold. Finally, we assign to each block b the label that received the highest cumulative confidence, $\max_l(C(b,l))$.

4. Building the *CADTalk* Dataset

While a few datasets of CAD programs have been recently introduced [43, 44], these programs do not include ground-truth semantic comments. We address this issue by introducing *CADTalk*, a dataset of OpenSCAD [30] programs enriched with part-based semantic comments (Figs. 1 and 5). We first describe how we collected and commented on the programs in our dataset, and then introduce the metrics we used to evaluate the quality of automatically generated comments against this benchmark.

4.1. Collecting CAD Programs

We consider two distinct sources of programs to comment on, each raising its specific challenges. On the one hand, *human-made programs* are often well-structured, with meaningful parts represented by independent code blocks. However, these programs exhibit a large diversity of instructions and program constructs, including macros, nested loops, and boolean operations to create intricate geometry, which can be difficult to reason about in the program domain. On the other hand, *machine-made* programs tend to obey a simplified CAD language with few instructions [17], resulting in a rather flat structure without obvious meaningful code blocks. Moreover, such programs often generate abstract geometry made of simple primitives (cuboids, ellipsoids), which can be difficult to reason about in the visual domain. We built our dataset to contain representative programs of both sources.

Human-made programs. We gathered a set of 45 OpenSCAD programs from online repositories [3, 5], or made by one of the authors. For each such program, we first identify each *commentable* code block as described in Sec. 3.3 (see Fig. 4). We then manually comment on each such block with a label indicating the semantic part of the shape that the code block corresponds to. When several irreducible blocks correspond to the same semantic part, we comment on each of them with the same label. We used ChatGPT-v4 [34] to obtain a list of semantic part labels given the category name of the object of interest.

We name the resulting set of semantically commented programs *CADTalk-Real*. While this set is small due to the difficulty of finding real programs and commenting on them by hand, Figs. 1 and 5 show that it includes diverse human-made and organic shapes of varying complexity.

Machine-made programs. Recent developments in shape analysis and program synthesis have enabled significant progress in generative models of CAD programs [39]. While the most recent models seek to capture high-level geometric structures [18, 19], others produce a flat list of geometric primitives [17, 41]. Since the human-made programs we have collected already exhibit complex structures, we focused the second track of our dataset on flat programs.

We rely on automatic methods that convert 3D shapes

Table 1. ***CADTalk* Statistics.** The number of programs, lines of code, and the number of parts are listed.

	#Programs	#Lines (min, median, max)	#Parts
CADTalk-Cube^L	1322	(21, 40, 66)	4,4,4,2
CADTalk-Cube^H	1322	(61, 72, 162)	4,4,4,2
CADTalk-Ellip^L	1322	(7, 115, 1077)	4,4,4,2
CADTalk-Ellip^H	1322	(27, 162, 1172)	4,4,4,2
CADTalk-Real	45	(28, 120, 381)	2-10

into cuboid [41] and ellipsoid [28] abstractions. By feeding these methods with semantically-labeled shapes from PartNet [32], we obtain programs formed as unions of cuboids or ellipsoids, each such primitive forming a code block associated with a semantic label. In the eventuality where a primitive covers several semantic parts of a shape, we associate that primitive with all the corresponding labels, each being treated as ground truth in our evaluation (i.e., either of the labels predicted is counted as correct). Consecutive blocks with the same labels can then be naturally grouped.

We followed this procedure to generate programs for four object categories (Airplane, Chair, Table, and Animal). As illustrated in Fig. 1, the cuboid abstractions typically only contain a few primitives, resulting in a coarse approximation of the shapes, while the ellipsoid abstractions better reproduce curved surfaces and details.

To further evaluate the impact of shape approximation, we provide for each of the two types of abstraction (cuboids and ellipsoids) two levels of details (low and high), resulting in four sets of program named *CADTalk-Cube*^{L}, *CADTalk-Cube*^{H}, *CADTalk-Ellip*^{L}, and *CADTalk-Ellip*^{H}, respectively. On the one hand, a high level of details results in long programs to comment on, where multiple primitives need to be grouped under the same part label. On the other hand, a low level of details results in approximate shapes that are harder to recognize (Fig. 12 in the supplementary).

Tab. 1 provides statistics of our dataset (number of programs, number of lines of code per program, number of semantic parts per program). In total, the dataset contains over 5300 commented CAD programs. The *CADTalk-Cube* and *CADTalk-Ellip* tracks allow to evaluate performance of commenting algorithms at a large scale, while the *CADTalk-Real* track provides a smaller albeit more diverse test set.

4.2. Evaluation Metrics

We propose two metrics to evaluate the performance of algorithms on the new task of commenting CAD programs. The first metric is block accuracy (B_{acc}), which calculates the successful rate of block-wise labeling, while the second metric is semantic IoU (S_{IoU}), which measures the Intersection-over-Union value per semantic label, averaged over all labels. Detailed formulations of the metrics can be found in the supplemental material.

Intuitively, block accuracy quantifies the general performance, while the semantic IoU considers the label distribution among blocks, making it sensitive to the long-tail problem where some labels only cover a few blocks.

Evaluation with synonyms. In practice, algorithms addressing the CAD program commenting problem may generate different labels than our ground truth annotations, albeit with a similar semantic meaning. We account for these synonyms in our evaluation by asking ChatGPT-v4 to give us, for a given list of predicted labels, its mapping to the list of ground truth labels (if any). We then apply the mapping before computing the metrics.

5. Experiments

We now describe our statistical and visual evaluations, we provide implementation details as supplemental material.

5.1. Results

Tab. 2 reports the quantitative evaluation of our *CADTalker* algorithm on the different tracks of our *CADTalk* dataset. For each track, we evaluate our algorithm when provided with either the ground-truth list of labels (GT Words) or the list suggested by ChatGPT (GPT Words). For each setting, we report both the block accuracy and the semantic IoU. Our pipeline is not restricted to GPT4. We have also tested with the open-source Llama2-70B (see supplementary).

Overall, our algorithm achieves similar results in both settings, demonstrating the effectiveness of synonym mapping and the applicability of our algorithm to real-world scenarios where ground truth labels are not available. The two metrics reach slightly higher values on cuboid than on ellipsoid abstractions, and on high level of details than on low level. Ellipsoid abstractions often contain overlapping primitives, which might be more difficult to label.

The real human-made programs in *CADTalk-Real* are more diverse in terms of program structure, shape geometry, and shape semantic granularity, leading to a 78.29% and 66.22% for block accuracy and semantic IoU, respectively. Our method sometimes mislabels parts that are spatially close and semantically related, like the steam dome vs. chimney in Fig. 1. Note that the ground truth labels we have specified in our dataset might be biased towards a specific granularity, which influences the metric (e.g., a coarse level of semantics is easier to predict). Fig. 5 illustrates typical visual results of our automatic commenting. More results and failure cases can be found in the supplementary.

5.2. Ablation Study

Table 3 compares several versions of our algorithm.

Multi-image generation. Our algorithm generates 4 realistic images for each of the 10 views of the shape. Reducing to one image per view decreases accuracy (Table 3, w/o MI), as the method is more sensitive to ambiguous shading, texture and other artifacts produced by ControlNet [47].

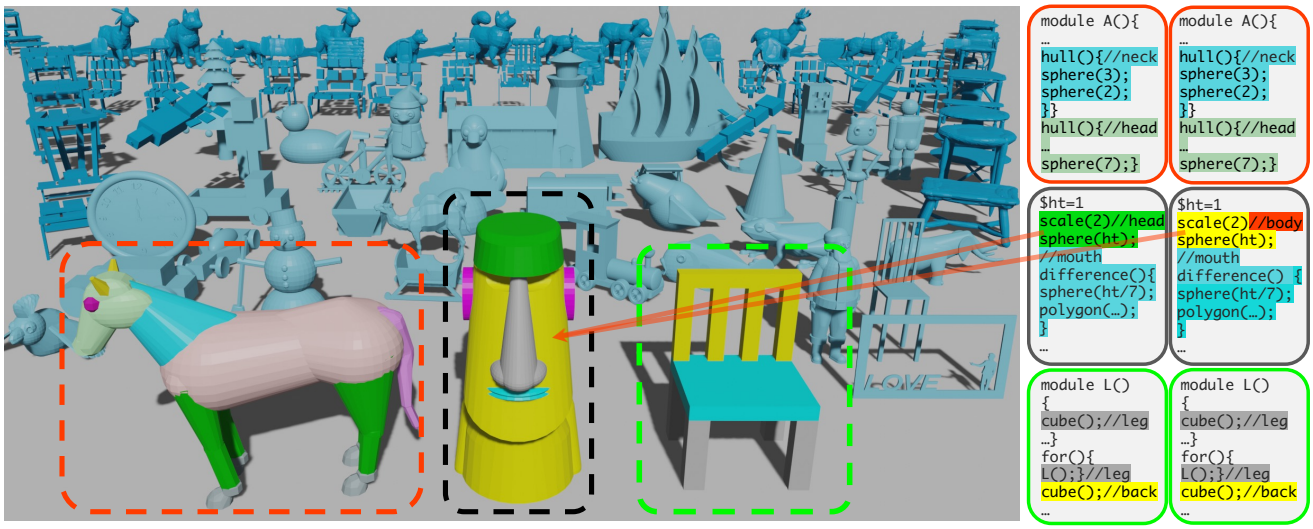


Figure 5. ***CADTalk* Dataset.** Example shapes from *CADTalk* (left) along with ground-truth (right) and predicted comments (far right). In these examples, our prediction matches the ground truth, except for the Moai sculpture where *CADTalker* labeled the "head" code block as "body". Machine-made shapes are rendered with dark blue and placed behind the human-made shapes rendered with light blue.

Table 2. Evaluation of our method on *CADTalk-Cube* and *CADTalk-Ellip* benchmarks. See Table 4 for comparison with PartSLIP [26].

Dataset	*CADTalk-Cube*								*CADTalk-Ellip*							
	CADTalk-Cube^H				*CADTalk-Cube^L*				*CADTalk-Ellip^H*				*CADTalk-Ellip^L*			
Input Text	GPT Words		GT Words		GPT Words		GT Words		GPT Words		GT Words		GPT Words		GT Words	
Metric	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}
Airplane (4 parts)	85.03	77.76	88.00	80.05	75.65	68.40	72.17	65.27	74.78	65.77	80.04	69.25	72.52	65.90	76.60	65.70
Chair (4 parts)	90.02	84.34	88.20	79.42	92.28	91.11	93.36	90.53	77.66	71.29	81.13	73.24	71.41	57.95	75.40	61.00
Table (2 parts)	90.88	86.33	90.91	85.64	95.76	93.50	90.90	85.89	83.06	74.71	83.56	74.60	81.21	75.76	79.02	72.38
Animal (4 parts)	89.07	82.55	86.15	77.79	89.03	85.68	88.70	84.07	91.28	83.14	89.42	81.70	91.90	86.08	85.68	74.81
Average	88.75	82.75	88.32	80.73	88.18	84.76	86.28	81.84	81.70	73.73	83.54	74.70	79.26	71.42	79.18	68.47

Table 3. Statical evaluation of our ablation study with block accuracy B_{acc}.

Dataset	*CADTalk-Cube*								*CADTalk-Ellip*							
	CADTalk-Cube^H				*CADTalk-Cube^L*				*CADTalk-Ellip^H*				*CADTalk-Ellip^L*			
Ab. Cond.	w/o MI	w/o CN	w/o SM	Full	w/o MI	w/o CN	w/o SM	Full	w/o MI	w/o CN	w/o SM	Full	w/o MI	w/o CN	w/o SM	Full
Airplane	76.93	37.02	28.47	**85.03**	65.87	26.41	25.61	**75.65**	70.91	25.90	32.84	**74.78**	68.67	25.92	24.83	**72.52**
Chair	73.77	59.70	35.48	**90.02**	88.48	38.50	21.28	**92.28**	73.44	70.62	33.59	**77.66**	67.32	60.93	24.82	**71.41**
Table	86.35	79.79	59.90	**90.88**	91.95	80.72	39.23	**95.76**	75.44	80.01	50.78	**83.06**	77.70	76.08	35.56	**81.21**
Animal	81.75	25.09	56.94	**89.07**	74.75	21.93	38.20	**89.03**	86.76	23.53	48.86	**91.28**	87.15	22.18	49.94	**91.90**
Average	79.70	50.40	45.20	**88.75**	80.26	41.89	31.08	**88.18**	76.64	50.02	41.52	**81.70**	75.21	46.28	33.79	**79.26**

Image-to-image translation. We leverage ControlNet [47] to turn our renderings of the shape into realistic images. Removing this component dramatically reduces accuracy (Tab. 3, w/o CN) due to the domain gap between our synthetic renderings and the photographs for which Grounded DINO and SAM have been trained.

Pixel-level segments. We employ an open-vocabulary detection method [27] to predict labeled bounding boxes in images, and we refine these boxes into pixel-level segments using SAM [21]. Tab. 3 (w/o SM) shows that accuracy drops significantly if we omit this last segmentation step, and instead rely on box-based IoU to evaluate Eq. 1. Bounding boxes only loosely delineate object parts, resulting in substantial noise in the voting process.

5.3. Comparison with PartSLIP [26]

Given that CAD programs are another type of 3D representation, our semantic program-commenting task can be considered a zero-shot, open-set 3D part segmentation problem. We compare our method with the state-of-the-art zero-shot 3D point cloud segmentation method PartSLIP [26].

Specifically, we convert *CADTalk* shapes into point

Table 4. Statistical Comparison with PartSLIP (PS) and its variants. Block accuracy B_{acc} is reported.

Dataset	*CADTalk-Cube*						*CADTalk-Ellip*					
	CADTalk-Cube^H			*CADTalk-Cube^L*			*CADTalk-Ellip^H*			*CADTalk-Ellip^L*		
Methods	PS	PS++	Ours	PS	PS++	Ours	PS	PS++	Ours	PS	PS++	Ours
Airplane	22.04	30.07	**85.03**	14.51	23.31	**75.65**	22.84	23.88	**74.78**	17.83	20.61	**72.52**
Chair	42.04	51.01	**90.02**	36.82	44.90	**92.28**	37.97	46.40	**77.66**	43.00	49.12	**71.41**
Table	17.83	56.79	**90.88**	16.23	58.07	**95.76**	20.55	61.96	**83.06**	17.56	58.01	**81.21**
Animal	36.76	55.64	**89.07**	38.26	54.40	**88.18**	23.33	42.32	**81.70**	18.48	35.48	**91.90**
Average	29.67	48.38	**88.75**	26.46	45.17	**87.97**	26.17	43.64	**79.30**	24.22	40.80	**79.26**

clouds and feed them to PartSLIP. Since PartSLIP outputs point-wise labels, we aggregate the point labels back to the program blocks based on block-point correspondence. Each block is assigned a label corresponding to most of the points it contains. Table 4 displays the statistical comparison, where the block accuracy is much lower than ours. We hypothesize that this performance drop is because PartSLIP relies on traditional rendering of the point cloud to perform visual reasoning, which results in non-realistic images in our context. Besides, PartSLIP assigns a null label to points that cannot be semantically recognized. Since null labels decrease our accuracy metrics, we also implemented a version that assigns a random label drawn from the list of ground-truth labels to each code block not recognized by PartSLIP (Tab. 4, PS++). While this variant achieves higher metric values, it remains inferior to ours. The same observation applies to *CADTalk-Real* (e.g., 78.29% vs. 18.06% for ours and PS in terms of B_{acc}); our prediction is better. Please refer to the supplemental.

5.4. Commenting on Machine-made Macros

While the machine-made programs in our dataset exhibit a flat structure, methods like ShapeCoder [19] can automatically discover abstractions within flat programs to form libraries of nested functions. We provide as supplemental materials an experiment on ShapeCoder programs, showing that while *CADTalker* produces comments that convey the semantic meaning of the functions, these functions sometimes mix several semantic parts since ShapeCoder created them solely based on geometry.

5.5. Semantic Commenting using ChatGPT

The key idea of *CADTalker* is to execute and render the CAD shape to cast program commenting as an image segmentation task. While our evaluation on *CADTalk* demonstrates the effectiveness of this image-based strategy, it can struggle in the presence of small parts and occlusions.

These limitations motivated us to experiment with a program-based strategy, for which visibility and appearance are irrelevant. Specifically, inspired by recent successes of few-shot training of LLMs for code commenting [2, 6], as well as by ongoing effort on leveraging LLMs to help designers in various tasks [29], we instructed ChatGPT-v4 to comment on a program given another program with comments as an example. We provide the results of this experiment as supplemental materials, which reveals that ChatGPT succeeds in commenting on programs that are similar to the example (same object category, same geometric primitives) but fails to generalize to new shapes and primitives.

6. Conclusion

In this paper, we have introduced the new problem of semantic commenting of CAD programs and proposed a novel method, *CADTalker*, to tackle this problem by combining program parsing with visual-semantic analysis offered by recent advances in foundational language and vision models. We have established an effective baseline for this new problem. Additionally, we have prepared a new benchmark dataset – *CADTalk* – to evaluate the performance of our algorithm and to facilitate future research in this direction.

Limitations. Our current algorithm cannot perform any non-trivial reordering or restructuring of input CAD programs. Hence, if an input program is badly written, instead of simply being badly commented, we cannot improve it to make it more readable. This leaves out an interesting optimization axis as even the exact same geometric shape can have many different programmatic realizations. Our method can only comment on instructions that produce geometry, not on instructions that remove geometry (which we aggregate within irreducible blocks). For example, if a shape is subtracted from another shape to form a hole, we cannot comment on the semantic meaning of that hole (as for the window of the cab of the train in Fig 1). A possible solution would be to explicitly render the geometry removed by the subtraction (i.e., for $A - B$, render $A \cap B$). Finally, the *CADTalk-Real* track is currently limited to 45 models, which is too little to serve as training data.

Future work. Our early experiments with ChatGPT (Sec 5.5) suggest that a multi-modality approach (e.g., taking as input tuples of programs, 2D renderings, and 3D shapes) is a promising direction for future work as it would allow reasoning on both the program structure and its visual output. Also, our ability to populate flat machine-generated programs with comments could benefit analysis of these programs, for instance, to account for both the geometry and semantics of the shape when searching for program abstractions [18, 19]. Finally, we would also like to assign meaningful labels to the program parameters to make it easier for human users to edit the programs (e.g., renaming the variable that controls how tall a chair back is to 'height'). Vision models capable of reasoning about *differences* between images [35] or semantic 3D features [11] might help.

Acknowledgments. We thank Algot Runeman for his OpenSCAD programs. CJ was supported by a startup grant from the School of Informatics, Bayes Seed funding, and gifts from Google Cloud research credits. NM was supported by Marie Skłodowska-Curie grant agreement No. 956585 and UCL AI Centre.

References

[1] Ahmed Abdelreheem, Ivan Skorokhodov, Maks Ovsjanikov, and Peter Wonka. Satr: Zero-shot semantic segmentation of 3d shapes. *arXiv preprint arXiv:2304.04909*, 2023. 3

[2] Toufique Ahmed and Premkumar Devanbu. Few-shot training llms for project-specific code-summarization. In *Proc. IEEE/ACM International Conference on Automated Software Engineering*, 2023. 3, 8, 1

[3] Algot Runeman. OpenSCAD 3D Central. http://runeman.org/3d/, 2023. 2, 5

[4] Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. In *International Conference on Learning Representations (ICLR)*, 2019. 3

[5] Felix W Baumann and Dieter Roller. Thingiverse: review and analysis of available files. *International Journal of Rapid Manufacturing*, 7(1):83–99, 2018. 2, 5

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 3, 8, 1

[7] Tianshi Cao, Karsten Kreis, Sanja Fidler, Nicholas Sharp, and Kangxue Yin. Texfusion: Synthesizing 3d textures with text-guided image diffusion models. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. 4

[8] D. Cascaval, M. Shalah, P. Quinn, R. Bodik, M. Agrawala, and A. Schulz. Differentiable 3d cad programs for bidirectional editing. *Computer Graphics Forum (Proc. EUROGRAPHICS)*, 41(2), 2022. 2

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. 3

[10] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 37(6):1–16, 2018. 2

[11] Niladri Shekhar Dutt, Sanjeev Muralikrishnan, and Niloy J. Mitra. Diffusion 3d features (diff3f): Decorating untextured shapes with distilled semantic features. In *CVPR*, 2024. 3, 8

[12] Github Authors. Lark - a parsing toolkit for Python. https://github.com/lark-parser/lark, 2017. 5

[13] Sonia Haiduc, Jairo Aponte, Laura Moreno, and Andrian Marcus. On the use of automated text summarization techniques for summarizing source code. In *Proc. Working Conference on Reverse Engineering*. IEEE Computer Society, 2010. 3

[14] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: A network with an edge. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 38(4), 2019. 3

[15] Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, (4):532–550, 1987. 4

[16] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *Proc. Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2016. 3

[17] R Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. Shapeassembly: Learning to generate programs for 3d shape structure synthesis. *ACM Transactions on Graphics (TOG)*, 39(6):1–20, 2020. 2, 5

[18] R Kenny Jones, David Charatan, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. Shapemod: macro operation discovery for 3d shape programs. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021. 5, 8

[19] R Kenny Jones, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *arXiv preprint arXiv:2305.05661*, 2023. 2, 5, 8, 1

[20] Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 3D shape segmentation with projective convolutional networks. In *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[21] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 3, 4, 7, 5

[22] Milin Kodnongbua, Benjamin T. Jones, Maaz Bin Safeer Ahmad, Vladimir G. Kim, and Adriana Schulz. Reparamcad: Zero-shot cad re-parameterization for interactive manipulation. In *SIGGRAPH Asia (Conference track)*, 2023. 2

[23] Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J. Mitra. Free2cad: Parsing freehand drawings into cad commands. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 41(4), 2022. 2

[24] Liunian Harold Li*, Pengchuan Zhang*, Haotian Zhang*, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu

Yuan, Lei Zhang, Jenq-Neng Hwang, Kai-Wei Chang, and Jianfeng Gao. Grounded language-image pre-training. In *Proc. IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 2022. 3

[25] Feng Liang, Bichen Wu, Xiaoliang Dai, Kunpeng Li, Yinan Zhao, Hang Zhang, Peizhao Zhang, Peter Vajda, and Diana Marculescu. Open-vocabulary semantic segmentation with mask-adapted clip. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7061–7070, 2023. 3

[26] Minghua Liu, Yinhao Zhu, Hong Cai, Shizhong Han, Zhan Ling, Fatih Porikli, and Hao Su. Partslip: Low-shot part segmentation for 3d point clouds via pretrained image-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21736–21746, 2023. 2, 3, 7

[27] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023. 3, 4, 7, 5

[28] Weixiao Liu, Yuwei Wu, Sipu Ruan, and Gregory S Chirikjian. Marching-primitives: Shape abstraction from signed distance function. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8771–8780, 2023. 2, 6

[29] Liane Makatura, Michael Foshey, Bohan Wang, Felix HähnLein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Spielberg, Crystal Elaine Owens, Peter Yichen Chen, Allan Zhao, Amy Zhu, Wil J Norton, Edward Gu, Joshua Jacob, Yifei Li, Adriana Schulz, and Wojciech Matusik. How can large language models help humans in design and manufacturing?, 2023. 8

[30] Marius Kintel. OpenSCAD. https://openscad.org/index.html, 2023. 4, 5

[31] Elie Michel and Tamy Boubekeur. Dag amendment for inverse control of parametric shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 40(4), 2021. 2

[32] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proc. IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pages 909–918, 2019. 6

[33] Chandrakana Nandi, Max Willsey, Adam Anderson, James R. Wilcox, Eva Darulova, Dan Grossman, and Zachary Tatlock. Synthesizing structured cad models with equality saturation and inverse transformations. In *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020. 2

[34] OpenAI. ChatGPT (v4, June 13 version) [Large language model]. https://chat.openai.com, 2023. 4, 5

[35] Dong Huk Park, Trevor Darrell, and Anna Rohrbach. Robust change captioning. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 8

[36] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proc. International Conference on Machine Learning*, pages 8748–8763, 2021. 3

[38] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes. In *ACM SIGGRAPH Conference Proceedings*, 2023. 4

[39] Daniel Ritchie, Paul Guerrero, R. Kenny Jones, Niloy J. Mitra, Adriana Schulz, Karl D. D. Willis, and Jiajun Wu. Neurosymbolic Models for Computer Graphics. *Computer Graphics Forum*, 2023. 2, 5

[40] Gopal Sharma, Kangxue Yin, Subhransu Maji, Evangelos Kalogerakis, Or Litany, and Sanja Fidler. Mvdecor: Multi-view dense correspondence learning for fine-grained 3d segmentation. In *ECCV*, 2022. 3

[41] Chun-Yu Sun, Qian-Fang Zou, Xin Tong, and Yang Liu. Learning adaptive hierarchical cuboid abstractions of 3d shape collections. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019. 2, 5, 6

[42] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 36(4), 2017. 3

[43] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 40(4), 2021. 2, 5

[44] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 2, 5

[45] Xiang Xu, Pradeep Kumar Jayaraman, Joseph G. Lambourne, Karl D.D. Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable cad model generation. In *Proc. International Conference on Machine Learning (ICML)*, 2023. 2

[46] Fenggen Yu, Zhiqin Chen, Manyi Li, Aditya Sanghi, Hooman Shayani, Ali Mahdavi-Amiri, and Hao Zhang. Capri-net: Learning compact cad shapes with adaptive primitive assembly. In *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[47] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3836–3847, 2023. 3, 4, 6, 7

CADTalk: An Algorithm and Benchmark for Semantic Commenting of CAD Programs

Supplementary Material

7. Additional Results

7.1. Commenting on ShapeCoder [19] Programs

While the machine-made programs in our dataset exhibit a flat structure, methods like ShapeCoder [19] can automatically discover abstractions within flat programs to form libraries of nested functions. We have tested *CADTalker* on the abstracted shape programs from ShapeCoder.

Data processing and running. Since ShapeCoder only provides a simple executor that produces line renderings (Fig. 6 (a)), we resort to OpenSCAD as an alternative executor to obtain 3D shapes suitable for depth map rendering. Specifically, as shown in Fig. 6, for each ShapeCoder program, we first use its default executor to transform the program into cuboid primitives represented by a set of parameters (e.g., height, width, and translation). We then translate these cuboid primitives into an OpenSCAD program, which can be executed to get the required depth map. After the translation, each line of the ShapeCoder program corresponds to one or more OpenSCAD code lines. We run *CADTalker* to generate the semantic comment for each line and aggregate these comments for each ShapeCoder line by a simple non-repetitive merging. We transfer the semantic comments back to the ShapeCoder program by exploiting the recorded program line and code block correspondence.

Results. Fig. 7 (a) shows typical results of our algorithm on programs produced by ShapeCoder. While our comments convey the semantic meaning of the ShapeCoder functions, they also reveal that because the ShapeCoder algorithm solely works on geometry, it produces functions that mix semantic parts (Fig. 7 (b)). This experiment suggests that automatic commenting could serve as a way to evaluate the semantic coherence of automatically generated code macros.

7.2. Semantic Commenting using ChatGPT

The key idea of the algorithm we have proposed – *CADTalker*– is to execute and render the CAD shape to cast program commenting as an image segmentation task. While our evaluation on *CADTalk* demonstrates the effectiveness of this image-based strategy, it has some limitations. First, object parts can be occluded in most of the views, and as such do not get labeled. Similarly, small parts that only cover a few pixels tend to be ignored. Second, while our use of image-to-image translation greatly reduces the domain gap between renderings and photographs, the images we obtain might still contain unrealistic details that are difficult to recognize.

Table 5. Different configurations when interacting with GPT-4 to comment on a cuboid airplane program. Each row (b-e) corresponds to a different commented example provided to GPT-4 for one-shot training.

Option	Teach	Example Program	$B_{acc}(\uparrow)$
a	✗	✗	31.88
b	✓	Airplane^{Cube} (partial)	52.5
c	✓	Airplane^{Ellip}	37.5
d	✓	Airplane^{Cube}	**87.5**
e	✓	Chair^{Cube}	44.37

These limitations motivated us to also experiment with a program-based strategy, for which visibility and appearance are irrelevant. Specifically, inspired by recent successes of few-shot training of LLMs for code commenting [2, 6], we instructed ChatGPT-v4 to comment on an airplane program from our *CADTalk-Cube* dataset. In addition to the program to be commented on, we also provided ChatGPT with the list of part names (i.e., 'body', 'wings', 'tail', and 'engine'), as illustrated in Fig. 8, top row.

Within this setup, we tested five different configurations of the commenting task, as listed Tab. 5 where the superscript indicates the source of the example program.

- Option (a): zero-shot prediction, *no* additional information is provided to ChatGPT.
- Option (b): one-shot prediction, the example is *incomplete* and comes from the same *airplane* category in *CADTalk-Cube*.
- Option (c): one-shot prediction, the example is *complete* but made of *different* primitives (i.e., ellipsoid) from *CADTalk-Ellip*.
- option (d): one-shot prediction, the example is *complete* and comes from the same *airplane* category in *CADTalk-Cube*.
- option (e): one-shot prediction, the example is *complete* but from the *chair* category in *CADTalk-Cube*.

In all cases, we shuffle the code blocks of both the task program and the example program to avoid any influence of ordering. This experiment reveals that, to our surprise, a single example is enough for ChatGPT to successfully comment programs that represent the same object category, with the same geometric primitives (configuration d). When asked to explain its answer, ChatGPT reported using the volume and relative position of the parts as evidence, such

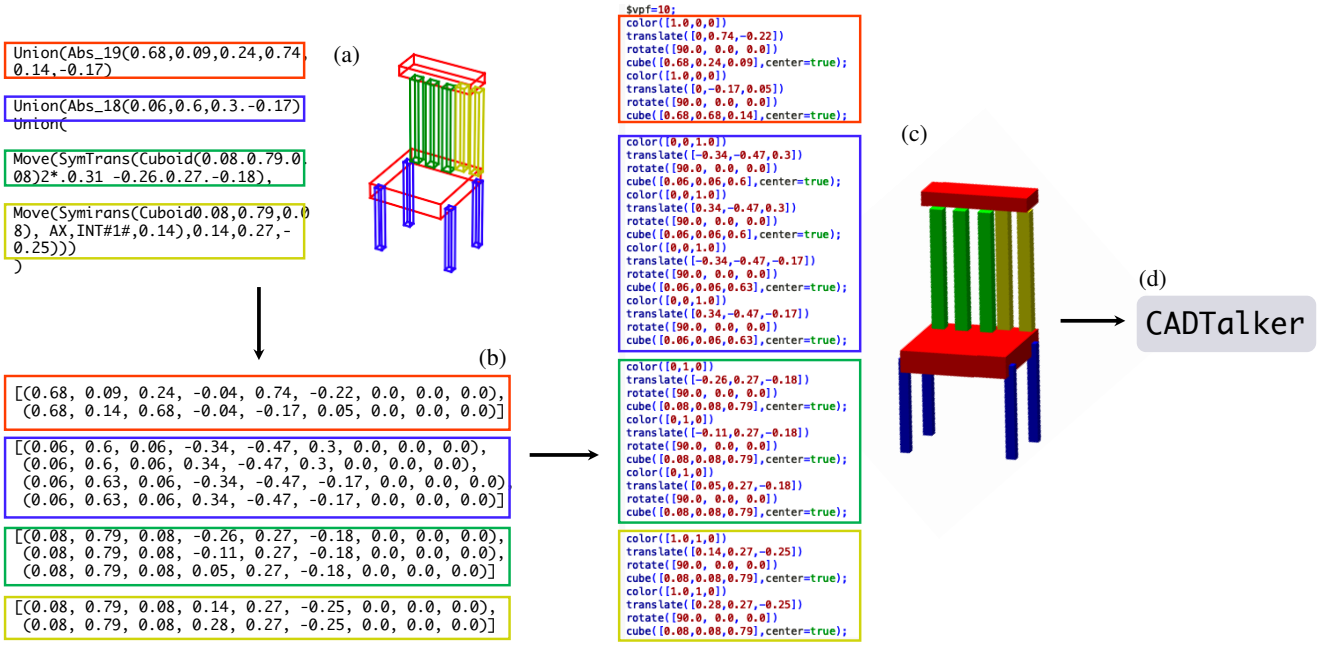


Figure 6. **ShapeCoder Program Processing**. Given a ShapeCoder program (a), we first execute the program to obtain all primitive parameters, i.e., a set of cubes represented by width, height, length, rotation, and translation (b). Then, we translate these primitives into an OpenSCAD program (c) and run our *CADTalker* (d). The colorful boxes indicate the line-parameter-code block correspondence.

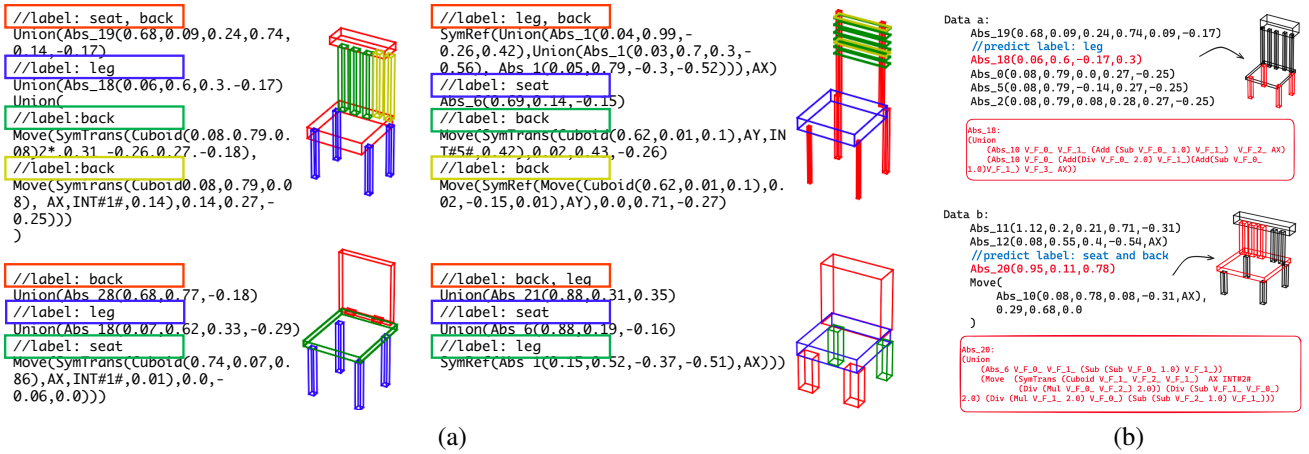


Figure 7. (a) Four typical commenting results on ShapeCoder programs with colored boxes indicating the comment-shape correspondence. (b) ShapeCoder [19] identifies redundancy in shape datasets to generate code macros (red blocks) that encapsulate common parts. While our approach produces descriptive comments for these macros (blue comments), the macros themselves do not always correspond to isolated semantic parts (bottom blue comments).

as the fact that the body should have the biggest volume, while the wings should be attached on the two sides of the body [1]. However, the other configurations (b,c,e) reveal that these spatial reasoning skills do not extend to examples that are incomplete, of another category, or made of different primitives. A small-scale statistical evaluation of these configurations with 10 testing examples is reported in Tab. 5, which is consistent with the above analysis and the visual results in Fig. 8, where the configuration (d) achieves the best result.

The full conversations with ChatGPT-v4 can be found in a separate file titled "GPT-Conversation.pdf" on the project page.

[1] GPT is trained to produce the next word given the prompt, we are not sure if it effectively used volume and positions to solve the task. It is an interesting research direction to reveal the mechanisms of GPT.

Table 6. Statistical Comparison with PartSLIP and its variants.

Dataset	*CADTalk-Cube*												*CADTalk-Ellip*											
	CADTalk-Cube^H						*CADTalk-Cube^L*						*CADTalk-Ellip^H*						*CADTalk-Ellip^L*					
Methods	PS		PS++		Ours		PS		PS++		Ours		PS		PS++		Ours		PS		PS++		Ours	
Metric	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}
Chair	62.81	59.74	64.51	59.35	**90.02**	**84.34**	74.24	60.29	74.90	59.60	**92.28**	**91.11**	64.92	57.46	64.94	57.20	**77.66**	**71.29**	66.86	52.23	67.54	51.99	**71.41**	**57.95**
Airplane	21.94	9.76	29.36	11.94	**85.03**	**77.76**	14.40	8.01	22.71	12.24	**75.65**	**68.40**	23.16	8.95	25.15	9.84	**74.78**	**65.77**	19.74	8.56	23.99	10.40	**72.52**	**69.90**
Animal	41.94	32.26	59.07	34.65	**89.07**	**82.75**	44.69	41.70	55.55	44.47	**89.03**	**85.68**	33.96	25.54	44.12	29.71	**91.28**	**83.14**	29.96	24.37	42.94	25.86	**91.90**	**86.08**
Table	40.90	33.43	64.98	49.64	**90.88**	**86.33**	37.60	27.55	65.61	49.50	**95.76**	**93.50**	38.38	35.02	63.34	45.09	**83.06**	**74.71**	43.05	32.04	61.38	53.66	**81.21**	**75.76**
Average	41.90	33.80	54.48	38.89	**88.75**	**82.80**	42.73	34.39	54.69	41.45	**88.18**	**84.67**	40.10	31.74	49.39	35.46	**81.69**	**73.73**	39.9	29.30	48.96	35.48	**79.26**	**72.42**

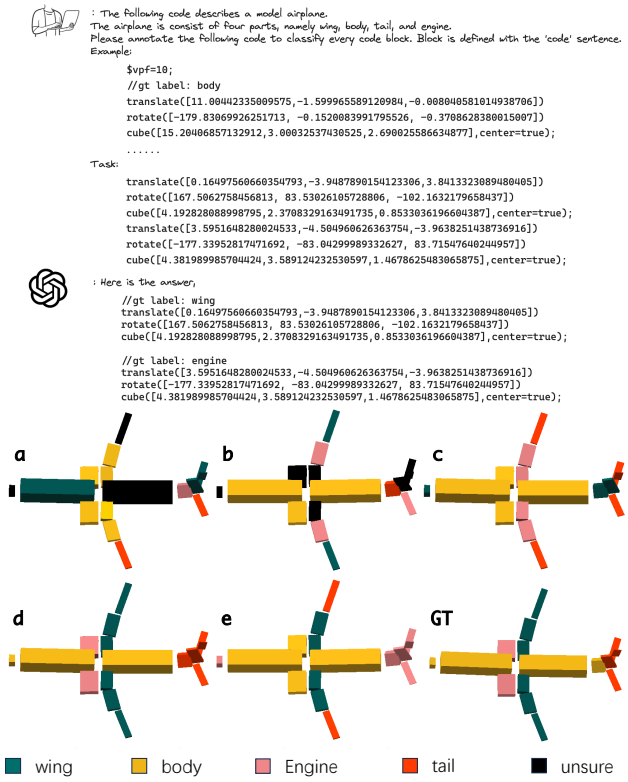


Figure 8. ChatGPT commenting results under different configurations.

7.3. Comparison with PartSLIP

In Sec. 5.3, we have described the comparison with PartSLIP regarding block accuracy. Here, we elaborate on the details and provide more statistical and visual results.

Data processing and running. Taking as input a dense and colored 3D point cloud and part names as prompts, PartSLIP predicts point-wise labels belonging to the part names. To compare, we first execute each program from *CADTalk* to obtain a 3D model and densely sample it with 200K points with normal and a uniform gray color (see Fig. 9, PartSLIP Input). As for the point-based rendering, we implemented a simple Phong shading[2] to produce the images fed to PartSLIP. In the sampling process, we record the point-block correspondence for label transferring using a similar binary mask based registration procedure as described in Sec. 3.3. This resulting point cloud is fed into PartSLIP to obtain point-wise labels. We then aggregate the point-wise labels of each commentable block by choosing the label with the highest number of votes and simply transfer the resulting label back to the shape program as the predicted comments.

Results. Full statistics with both the block accuracy (B_{acc}) and the semantic IoU (S_{IoU}) are shown in Tab. 6, where we obtain far better results compared with PartSLIP and PartSLIP++. As for the human-made program, the block accuracy for PartSLIP, PartSLIP++, and ours are 38.17% vs. 39.24% vs. 78.29%, while the semantic IoUs are 27.25%, and 27.73%, and 66.22%, respectively. Visual Results can be found in Fig. 9. PartSLIP fails this zero-shot point cloud segmentation task on both machine-made and human-made programs in our context. This is mainly attributed to PartSLIP's strong dependency on point clouds that incorporate realistic colors, a feature frequently absent in program representations. This failure is further evidenced in our ablation study, wherein the exclusion of ControlNet (w/o CN) results in notably reduced evaluation metrics.

7.4. Comparison with SATR

As discussed in Sec. 2 and Sec. 5.3, our task can be considered as a zero-shot, open-set 3D part segmentation problem. Other than PartSLIP, we preliminarily compare our method with SATR [1], the state-of-the-art zero-shot 3D mesh segmentation method. Qualitative results are shown in Fig. 10, where mesh segmentations are competitive on the realistic horse, but SATR struggles on the more abstracted Moai sculpture and fails on the abstracted airplane. The reason is the gap between the rendered images from

[2]The original PartSLIP code for point-based rendering does not apply any shading because it assumes that the input point cloud is a 3D colored scan. We replaced that code with our simple Phong shading. Some numbers reported in Tab. 4 of our submission were computed with the original rendering, which resulted in a lower performance. Nevertheless, even with better shading, PartSLIP's results are far inferior to ours. We will revise the numbers upon acceptance.

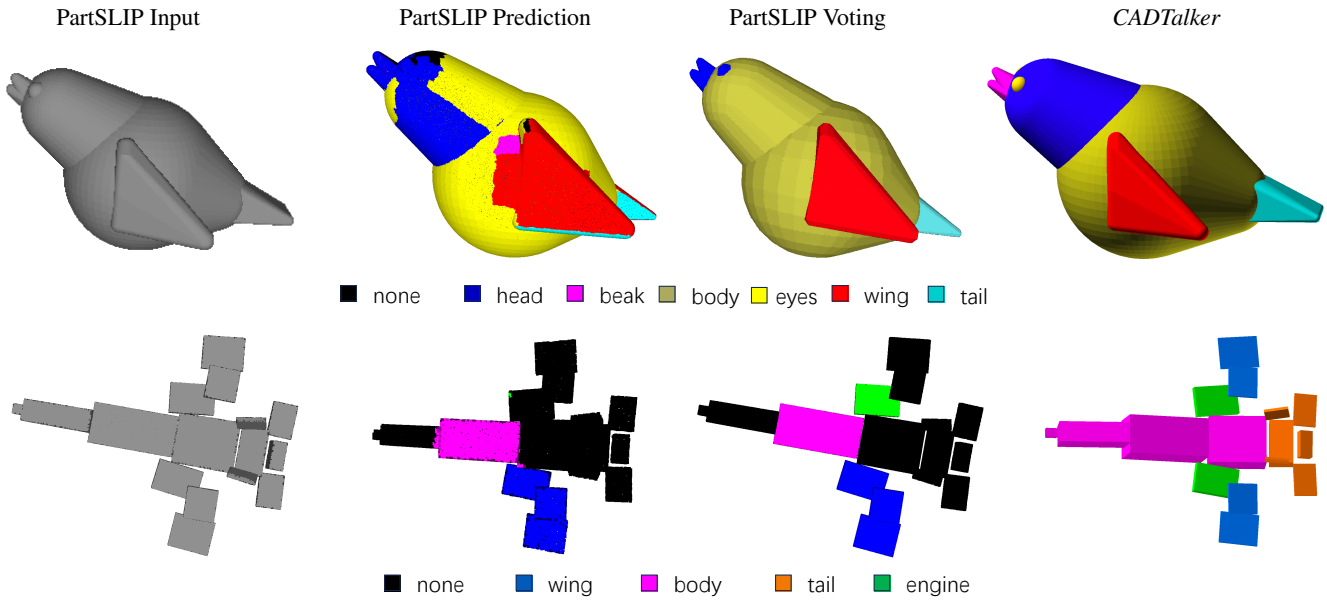


Figure 9. **Visual Comparison with PartSLIP**. Due to the absence of realistic colors, the raw prediction of the per-point label is noisy, leaning toward missing many points (the black color). After the label aggregation, errors are still obvious, e.g., the tail and most of the body of the airplane are mislabeled, while the head, beak, and eyes of the bird are totally wrong.

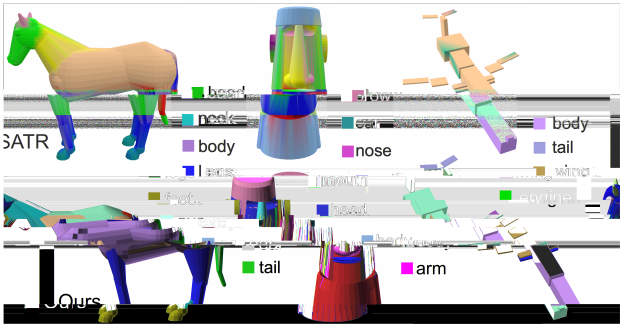


Figure 10. **Visual Comparison with SATR**. The first row shows the results from SATR, while our results are in the second row.

abstracted shapes and the photographs used for training the large image-language model, while ControlNet in our pipeline solves this problem effectively.

7.5. OpenLLM Model Test

Our pipeline is highly modular and not restricted to GPT4, we thus test our algorithm with the open-source Llama2-70B model. Statistical results are displayed in Table 7, where performance degradation is observed. For example, with Llama2-70B, the B_{acc} is dropped from 88.75% to 82.96% and S_{IoU} is dropped from 82.75% to 75.43% on *CADTalk-Cube*^{H} programs. As for the real human-made programs in CADTalk-Real, Llama2-70B achieves 70.88% and 57.97% for block accuracy and semantic IoU, which are reduced by 7.4% and 8.3% compared with GPT-4 (i.e., 78.29% and 66.22%, respectively). Once a more powerful LLM is available, our method can enjoy the improvement without any special tunning.

7.6. Additional Commenting Results

Typical commenting results can be seen on the accompanying webpage with highlighted code and block animations, and more commenting results from all data tracks in *CADTalk* can be found in a separate file titled "Commenting-Results.pdf" on the project page. In the following, we introduce typical failure cases.

Failure cases. In Fig. 11, we illustrate typical failure cases of our method, which are mainly inherited from foundational vision-language models, i.e., ControlNet may ignore fine details of the input depth map or generate totally unrecognizable images, and Grounding DINO may mislabel parts that can be seen clearly in the image.

To address these issues, potential solutions include a) utilizing stronger vision-language models with enhanced conditional generation ability, and more robust detection ability and b) implementing an image discriminator to exclude problematic images, which we leave for future work.

8. Method Details

8.1. Implementation Details

For depth map processing, we use morphological closing [15] with varied configurations. Specifically, we apply 5 iterations of closing for the abstract shapes of *CADTalk-Cube*^{H} and *CADTalk-Ellip*^{H}, 3 iterations for *CADTalk-Cube*^{L} and *CADTalk-Ellip*^{L}, and 1 iteration for *CADTalk-Real*, using a 3×3 structuring element. When using ControlNet [47], we set the control strength to 1.0, DDIM sam-

Table 7. Comparison between GPT words and LLAMA2 words on the full dataset.

Dataset	*CADTalk-Cube*								*CADTalk-Ellip*							
	CADTalk-Cube^H				*CADTalk-Cube*^{L}				*CADTalk-Ellip*^{H}				*CADTalk-Ellip*^{L}			
Input Text	GPT Words		LLAMA2 Words		GPT Words		LLAMA2 Words		GPT Words		LLAMA2 Words		GPT Words		LLAMA2 Words	
Metric	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}	B_{acc}	S_{IoU}
Airplane (4 parts)	85.03	77.76	85.71	78.28	75.65	68.40	77.32	71.51	74.78	65.77	77.43	71.19	72.52	65.90	75.90	70.66
Chair (4 parts)	90.02	84.34	87.18	77.33	92.28	91.11	91.47	88.11	77.66	71.29	75.49	68.56	71.41	57.95	68.99	55.30
Table (2 parts)	90.88	86.33	80.71	73.47	95.76	93.50	88.19	79.96	83.06	74.71	74.35	61.27	81.21	75.76	78.23	67.11
Animal (4 parts)	89.07	82.55	78.26	72.65	89.03	85.68	86.82	86.46	91.28	83.14	75.19	74.85	91.90	86.08	71.05	70.29
Average	88.75	82.75	82.96	75.43	88.18	84.76	85.95	81.51	81.70	73.73	75.62	68.97	79.26	71.42	74.54	65.84

(a) (b) (c)

Figure 11. **Failure Cases**. (a) ControlNet fails to generate scarf tassels. (b) ControlNet generates an unexpected image given a turkey depth map and keyword, as if it confused "turkey" (bird) with "Turkey" (country). (c) Grounding DINO wrongly predicts the broom to be 'head'.

pling steps to 20, the image instance number to 4, and the image resolution to 512×512. We use a simple text prompt template – "[CateName], realistic" for ControlNet, where [CateName] is the category name, e.g., Chair. We employ default parameter configurations from Grounding DINO [27] and SAM [21] without additional adjustments or tuning. For our voting scheme, we render depth maps from 10 viewpoints that are evenly distributed around a circular path centering on the object's up axis and maintaining an elevation angle of 55 degrees above the object. When filling in the cumulative confidence score, we progressively adjust the filtering threshold in the aforementioned three steps (Sec. 3.3), setting it at 0.001, 0.01, and 0.02, respectively.

Running Time. All experiments were conducted with a single RTX3090 GPU. Using our unoptimized code, for a program with 200 lines, the overall running time is around 6mins, distributed as 0.2% for program parsing, 1.1% for depth images rendering, 85.1% for ControlNet, 0.7% for prompt querying, 12.5% for DINO+SAM and 0.3% for voting.

8.2. Program Parsing

In Sec. 3.3, we introduced program parsing that produces an Abstract Syntax Tree (AST), laying the foundation for our commenting task. To do this, we exploit Lark [12] to conduct lexical and syntax analysis following the OpenSCAD grammar, and the analysis procedure generates an analysis tree, which is equivalent to the original program and wherein all the operation information is stored in the node and the code structure is maintained in the tree structure. Then, we construct the AST by traversing the analysis tree, and choose the required information, i.e., node type and line number, from the tree node. Example AST of a simple program is shown in Fig. 14, and more trees of programs in *CADTalk* can be found in the file titled "AST.pdf" on the project page.

9. *CADTalk* Dataset

9.1. Dataset Overview

To facilitate evaluation and foster future research on the semantic CAD program commenting task, we have introduced a new benchmark – *CADTalk*, a dataset of OpenSCAD programs enriched with part-based semantic comments. Tab. 8 shows detailed statistics per category of each data track, including the number of code lines, and the number of parts.

We considered two distinct sources of programs, i.e., human-made and machine-made programs, in our dataset. Since it is difficult to find and manually comment on real shape programs, we only gathered 45 such programs with rich shape and program diversity and we plan to keep col-

lecting more in the future. For machine-made programs, we rely on automatic methods that convert 3D shapes into cuboid [41] and ellipsoid [28]. One feature of this data track is the two levels of details of the programs, where the ones with a high level of detail reconstruct the shape well but have more lines to comment on, while the others with a low level of detail are harder to recognize due to the abstraction. See Fig. 12 for an example.

Figure 12. **Shape abstraction levels.** A chair in *CADTalk-Ellip* with different numbers of ellipsoids.

Table 8. **Detailed *CADTalk* Statistics.** The number of programs, lines of code, and the number of parts per category for each data track are listed.

	Category	#Programs	#Lines (min, median, max)	#Parts
CADTalk-Cube^L	airplane	400	(40, 40, 40)	4
	chair	400	(66, 66, 66)	4
	table	400	(21, 21, 21)	2
	animal	122	(40, 40, 40)	4
CADTalk-Cube^H	airplane	400	(72, 72, 72)	4
	chair	400	(162, 162, 162)	4
	table	400	(61, 61,61)	2
	animal	122	(72, 72, 72)	4
CADTalk-Ellip^L	airplane	400	(37, 100, 242)	4
	chair	400	(27, 147, 672)	4
	table	400	(7, 101, 1077)	2
	animal	122	(62, 112, 166)	4
CADTalk-Ellip^H	airplane	400	(32, 163, 237)	4
	chair	400	(57, 261, 842)	4
	table	400	(27, 178, 1172)	2
	animal	122	(27, 152, 245)	4
CADTalk-Real	real	45	(28, 120, 381)	2-10

9.2. Evaluation Metrics

We have proposed two metrics to evaluate the performance of algorithms on the new task of commenting CAD programs. In the following, we introduce the formulations to calculate them.

- *Block accuracy* is the block-wise labeling accuracy, defined as:

$$B_{acc} = \frac{m}{n}, \tag{2}$$

where m counts the number of blocks that get the correct label and n is the total number of blocks.

- *Semantic IoU* measures the Intersection-over-Union value per semantic label, averaged over all labels:

$$S_{IoU} = \frac{1}{K} \sum_k \frac{\{l_k\} \cap \{l_k^*\}}{\{l_k\} \cup \{l_k^*\}}, \tag{3}$$

where K is the number of labels, $\{l_k\}$ is the set of code blocks predicted to be of the k^{th} label, $\{l_k^*\}$ is the set of code blocks with the k^{th} label as ground truth.

9.3. Machine-made Program Processing

Given machine-generated shape primitives of ShapeNet models, we turn them into OpenSCAD programs and then conduct automatic labeling and manual refinement.

Program Translation. Given the cube or ellipsoid primitives represented by corresponding parameters, we trivially translate these primitives into OpenSCAD cube or ellipsoid primitives, following the same procedure as described in Fig. 6. Specifically, we translate a cube represented by its eight corners into the native cube primitive in OpenSCAD, while we translate an ellipsoid presented by its semi-axe lengths, rotation, and translation parameters into the native ellipsoid primitive in OpenSCAD.

Automatic Labels Transferring. Since cubes or ellipsoids are generated based on 3D models from PartNet, the existing part labels in PartNet can be utilized for part label assignment. Specifically, given a shape program, we first convert the corresponding PartNet shape into a point cloud with per-point labels. We then compare the part shape generated by each code block to the labeled point cloud by checking the IoU, and obtain the corresponding part label by maximum voting. For a part, e.g., the airplane engine, it may occupy both the wing and engine areas, we thus keep all valid labels in the voting.

Label Refinement with a Developed UI. For further refinement of the automatically generated labels, we also developed an interactive UI (Fig. 13) to directly review and adjust labeled programs in *CADTalk* by simple mouse clicking and keyboard hitting.

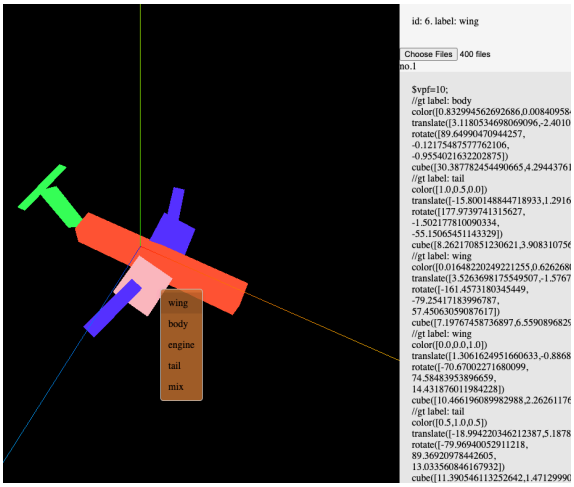


Figure 13. **User Interface.** The interface enables users to efficiently go through programs and adjust labels.

Figure 14. Abstracted Syntax Tree (AST). Each node in the AST maintains the operation type and the corresponding line number for pixel-block registration.