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.

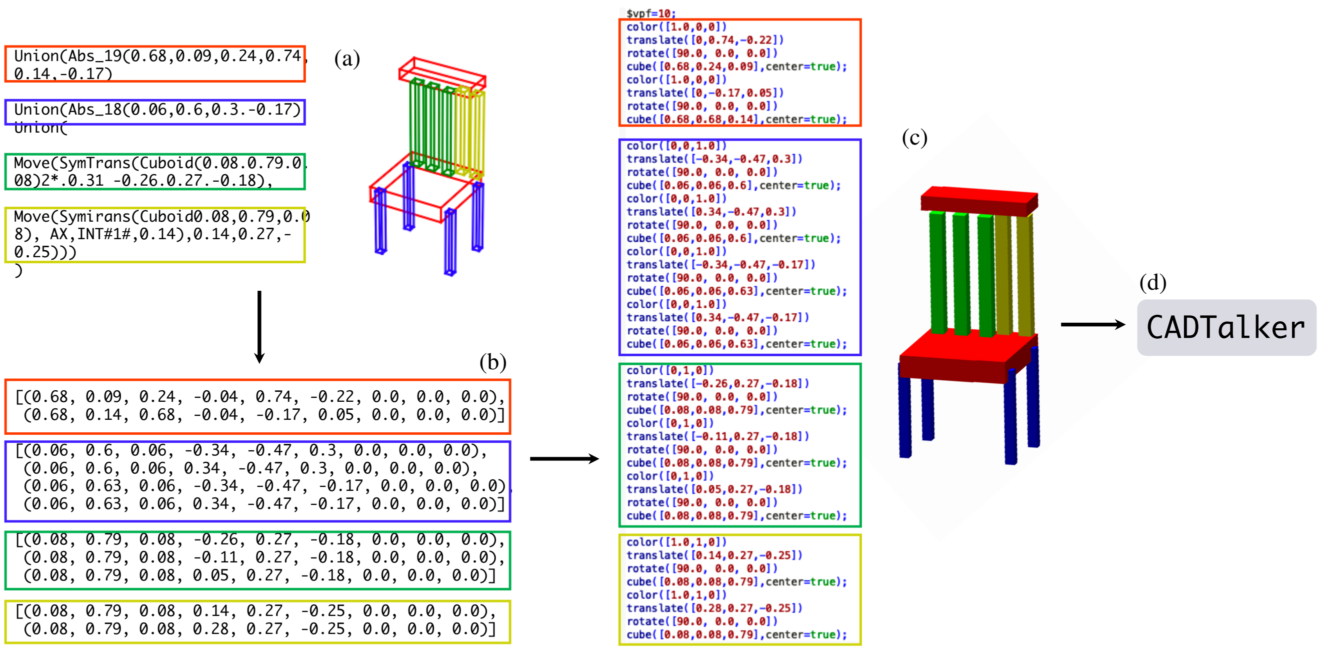
Option	Teach	Example Program	$B_{acc}(\uparrow)$
a	✗	✗	31.88
b	✓	Airplane^{Cube} (partial)	52.5
c	✓	Airplane^{Ellip}	37.5
d	✓	Airplane^{Cube}	**87.5**
e	✓	Chair^{Cube}	44.37

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



```
Union(Abs_19(0.68,0.09,0.24,0.74,
0.14,-0.17)

Union(Abs_18(0.06,0.6,0.3.-0.17)
Union(

Move(SymTrans(Cuboid(0.08.0.79.0.
08)2*.0.31 -0.26.0.27.-0.18),

Move(Symirans(Cuboid0.08,0.79,0.0
8), AX,INT#1#,0.14),0.14,0.27,-
0.25)))
)
```

```
[(0.68, 0.09, 0.24, -0.04, 0.74, -0.22, 0.0, 0.0, 0.0),
 (0.68, 0.14, 0.68, -0.04, -0.17, 0.05, 0.0, 0.0, 0.0)]
[(0.06, 0.6, 0.06, -0.34, -0.47, 0.3, 0.0, 0.0, 0.0),
 (0.06, 0.6, 0.06, 0.34, -0.47, 0.3, 0.0, 0.0, 0.0),
 (0.06, 0.63, 0.06, -0.34, -0.47, -0.17, 0.0, 0.0, 0.0),
 (0.06, 0.63, 0.06, 0.34, -0.47, -0.17, 0.0, 0.0, 0.0)]
[(0.08, 0.79, 0.08, -0.26, 0.27, -0.18, 0.0, 0.0, 0.0),
 (0.08, 0.79, 0.08, -0.11, 0.27, -0.18, 0.0, 0.0, 0.0),
 (0.08, 0.79, 0.08, 0.05, 0.27, -0.18, 0.0, 0.0, 0.0)]
[(0.08, 0.79, 0.08, 0.14, 0.27, -0.25, 0.0, 0.0, 0.0),
 (0.08, 0.79, 0.08, 0.28, 0.27, -0.25, 0.0, 0.0, 0.0)]
```

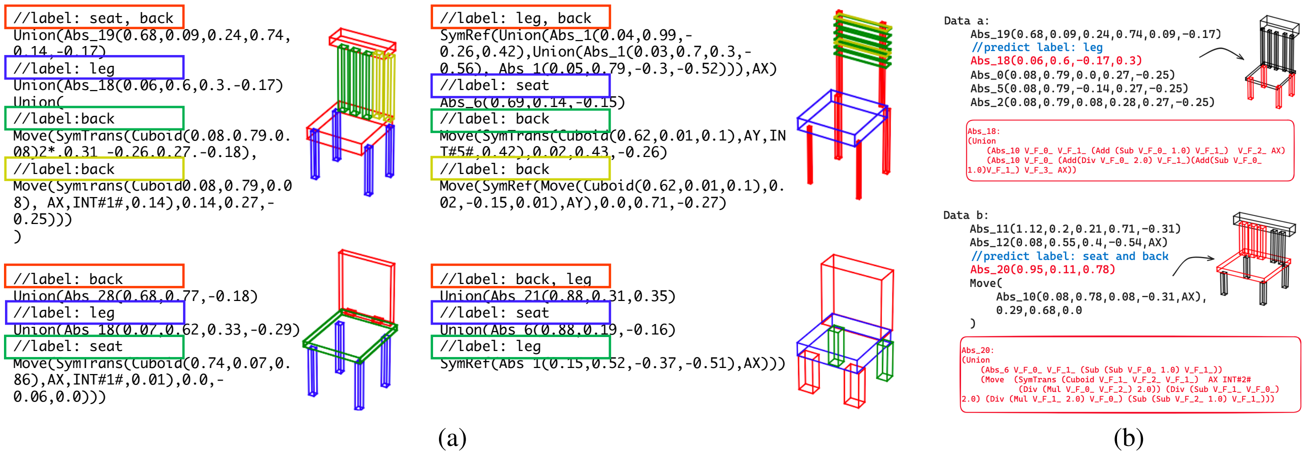
```
$vpf=10;
color([1.0,0,0])
translate([0,0.74,-0.22])
rotate([90.0, 0.0, 0.0])
cube([0.68,0.24,0.09],center=true);
color([1.0,0,0])
translate([0,-0.17,0.05])
rotate([90.0, 0.0, 0.0])
cube([0.68,0.68,0.14],center=true);

color([0,0,1.0])
translate([-0.34,-0.47,0.3])
rotate([90.0, 0.0, 0.0])
cube([0.06,0.06,0.6],center=true);
color([0,0,1.0])
translate([0.34,-0.47,0.3])
rotate([90.0, 0.0, 0.0])
cube([0.06,0.06,0.6],center=true);
color([0,0,1.0])
translate([-0.34,-0.47,-0.17])
rotate([90.0, 0.0, 0.0])
cube([0.06,0.06,0.63],center=true);
color([0,0,1.0])
translate([0.34,-0.47,-0.17])
rotate([90.0, 0.0, 0.0])
cube([0.06,0.06,0.63],center=true);

color([0,1,0])
translate([-0.26,0.27,-0.18])
rotate([90.0, 0.0, 0.0])
cube([0.08,0.08,0.79],center=true);
color([0,1,0])
translate([-0.11,0.27,-0.18])
rotate([90.0, 0.0, 0.0])
cube([0.08,0.08,0.79],center=true);
color([0,1,0])
translate([0.05,0.27,-0.18])
rotate([90.0, 0.0, 0.0])
cube([0.08,0.08,0.79],center=true);

color([1.0,1,0])
translate([0.14,0.27,-0.25])
rotate([90.0, 0.0, 0.0])
cube([0.08,0.08,0.79],center=true);
color([1.0,1,0])
translate([0.28,0.27,-0.25])
rotate([90.0, 0.0, 0.0])
cube([0.08,0.08,0.79],center=true);
```

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.



```
//label: seat, back
Union(Abs_19(0.68,0.09,0.24,0.74,
0.14,-0.17)
//label: leg
Union(Abs_18(0.06,0.6,0.3.-0.17)
Union(
//label:back
Move(SymTrans(Cuboid(0.08.0.79.0.
08)2*.0.31 -0.26.0.27.-0.18),
//label:back
Move(Symirans(Cuboid0.08,0.79,0.0
8), AX,INT#1#,0.14),0.14,0.27,-
0.25)))
)
```

```
//label: leg, back
SymRef(Union(Abs_1(0.04,0.99,-
0.26,0.42),Union(Abs_1(0.03,0.7,0.3,-
0.56), Abs_1(0.05,0.79,-0.3,-0.52))),AX)
//label: seat
Abs_6(0.69,0.14,-0.15)
//label: back
Move(SymTrans(Cuboid(0.62,0.01,0.1),AY,IN
T#5#,0.42),0.02,0.43,-0.26)
//label: back
Move(SymRef(Move(Cuboid(0.62,0.01,0.1),0.
02,-0.15,0.01),AY),0.0,0.71,-0.27)
```

```
//label: back
Union(Abs_28(0.68,0.77,-0.18)
//label: leg
Union(Abs_18(0.07,0.62,0.33,-0.29)
//label: seat
Move(SymTrans(Cuboid(0.74,0.07,0.
86),AX,INT#1#,0.01),0.0,-
0.06,0.0)))
```

```
//label: back, leg
Union(Abs_21(0.88,0.31,0.35)
//label: seat
Union(Abs_6(0.88,0.19,-0.16)
//label: leg
SymRef(Abs_1(0.15,0.52,-0.37,-0.51),AX)))
```

```
Data a:
    Abs_19(0.68,0.09,0.24,0.74,0.09,-0.17)
    //predict label: leg
    Abs_18(0.06,0.6,-0.17,0.3)
    Abs_0(0.08,0.79,0.0,0.27,-0.25)
    Abs_5(0.08,0.79,-0.14,0.27,-0.25)
    Abs_2(0.08,0.79,0.08,0.28,0.27,-0.25)

Abs_18:
(Union
    (Abs_10 V_F_0_ V_F_1_ (Add (Sub V_F_0_ 1.0) V_F_1_)  V_F_2_ AX)
    (Abs_10 V_F_0_ (Add(Div V_F_0_ 2.0) V_F_1_)(Add(Sub V_F_0_
1.0)V_F_1_) V_F_3_ AX))

Data b:
    Abs_11(1.12,0.2,0.21,0.71,-0.31)
    Abs_12(0.08,0.55,0.4,-0.54,AX)
    //predict label: seat and back
    Abs_20(0.95,0.11,0.78)
    Move(
        Abs_10(0.08,0.78,0.08,-0.31,AX),
        0.29,0.68,0.0
    )

Abs_20:
(Union
    (Abs_6 V_F_0_ V_F_1_ (Sub (Sub V_F_0_ 1.0) V_F_1_))
    (Move  (SymTrans (Cuboid V_F_1_ V_F_2_ V_F_1_)  AX INT#2#
            (Div (Mul V_F_0_ V_F_2_) 2.0)) (Div (Sub V_F_1_ V_F_0_)
2.0) (Div (Mul V_F_1_ 2.0) V_F_0_) (Sub (Sub V_F_2_ 1.0) V_F_1_)))
```

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).

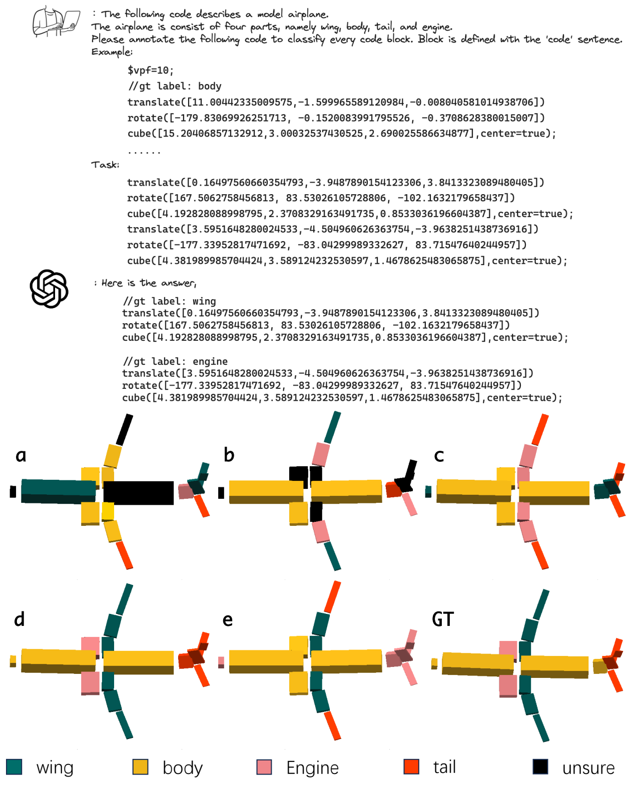
as the fact that the body should have the biggest volume, while the wings should be attached on the two sides of the body [1]. However, the other configurations (b,c,e) reveal that these spatial reasoning skills do not extend to examples that are incomplete, of another category, or made of different 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**


: The following code describes a model airplane.
The airplane is consist of four parts, namely wing, body, tail, and engine.
Please annotate the following code to classify every code block. Block is defined with the 'code' sentence.
Example:

```
        $vpf=10;
        //gt label: body
        translate([11.00442335009575,-1.599965589120984,-0.008040581014938706])
        rotate([-179.83069926251713, -0.1520083991795526, -0.3708628380015007])
        cube([15.20406857132912,3.00032537430525,2.690025586634877],center=true);
        ......
```

Task:

```
        translate([0.16497560660354793,-3.9487890154123306,3.8413323089480405])
        rotate([167.5062758456813, 83.53026105728806, -102.1632179658437])
        cube([4.192828088998795,2.3708329163491735,0.8533036196604387],center=true);
        translate([3.5951648280024533,-4.504960626363754,-3.9638251438736916])
        rotate([-177.33952817471692, -83.04299989332627, 83.71547640244957])
        cube([4.381989985704424,3.589124232530597,1.4678625483065875],center=true);
```

: Here is the answer,

```
       //gt label: wing
       translate([0.16497560660354793,-3.9487890154123306,3.8413323089480405])
       rotate([167.5062758456813, 83.53026105728806, -102.1632179658437])
       cube([4.192828088998795,2.3708329163491735,0.8533036196604387],center=true);

       //gt label: engine
       translate([3.5951648280024533,-4.504960626363754,-3.9638251438736916])
       rotate([-177.33952817471692, -83.04299989332627, 83.71547640244957])
       cube([4.381989985704424,3.589124232530597,1.4678625483065875],center=true);
```

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.

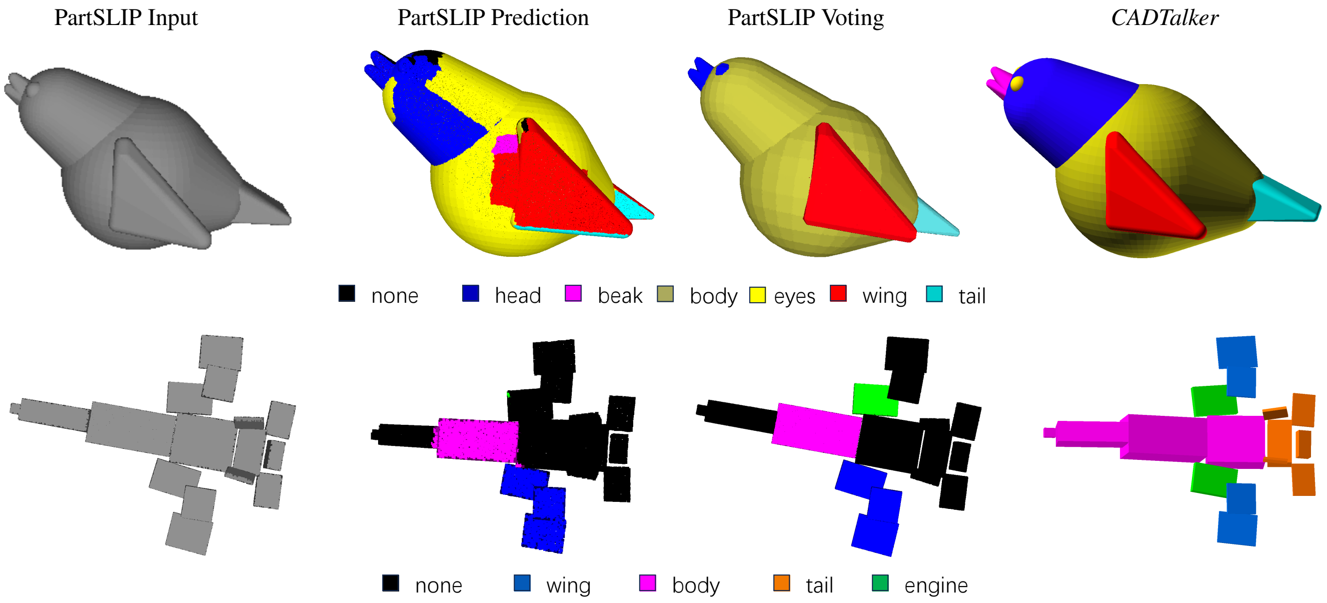


Figure 9. **Visual Comparison with PartSLIP**. Due to the absence of realistic colors, the raw prediction of the per-point label is noisy, leaning toward missing many points (the black color). After the label aggregation, errors are still obvious, e.g., the tail and most of the body of the airplane are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

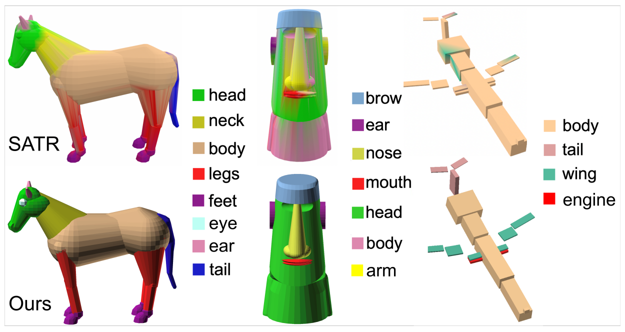


Figure 10. **Visual Comparison with SATR**. The first row shows the results from SATR, while our results are in the second row.

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.

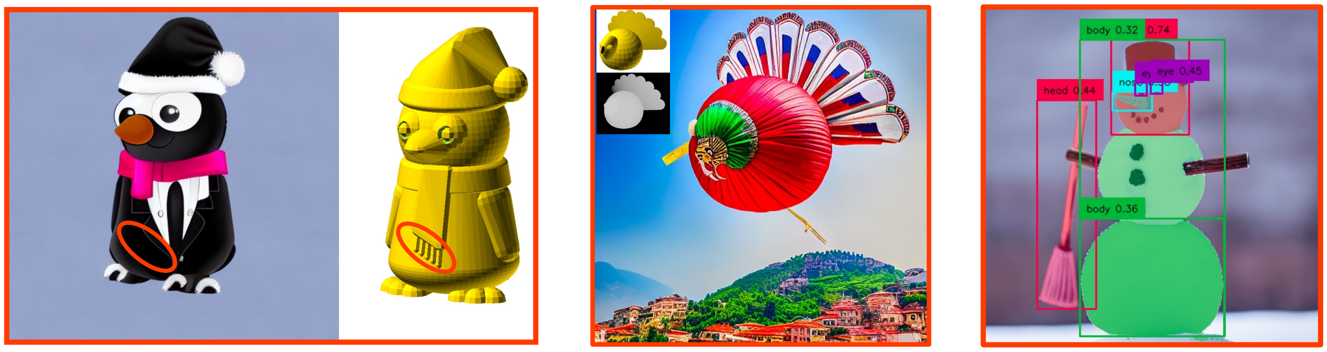
8. Method Details

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 collecting 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.

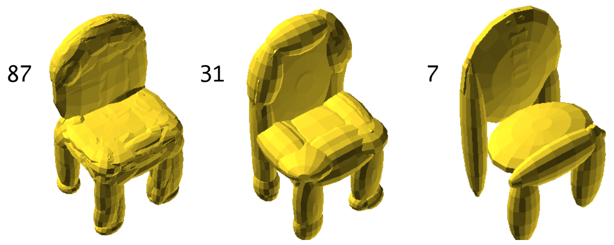


Figure 12. **Shape abstraction levels.** A chair in *CADTalk-Ellip* with different numbers of ellipsoids.

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.

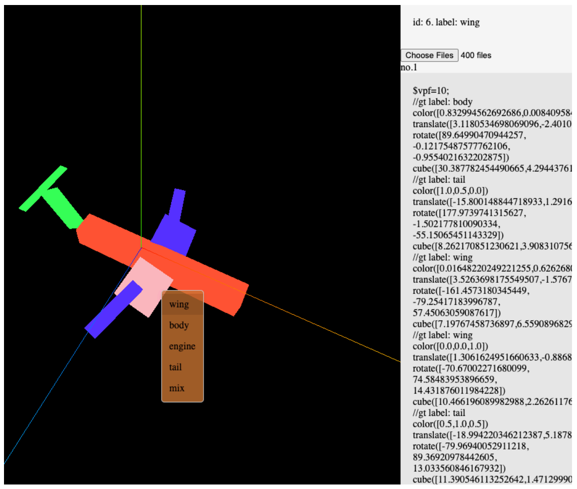
9.3. Machine-made Program Processing

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.

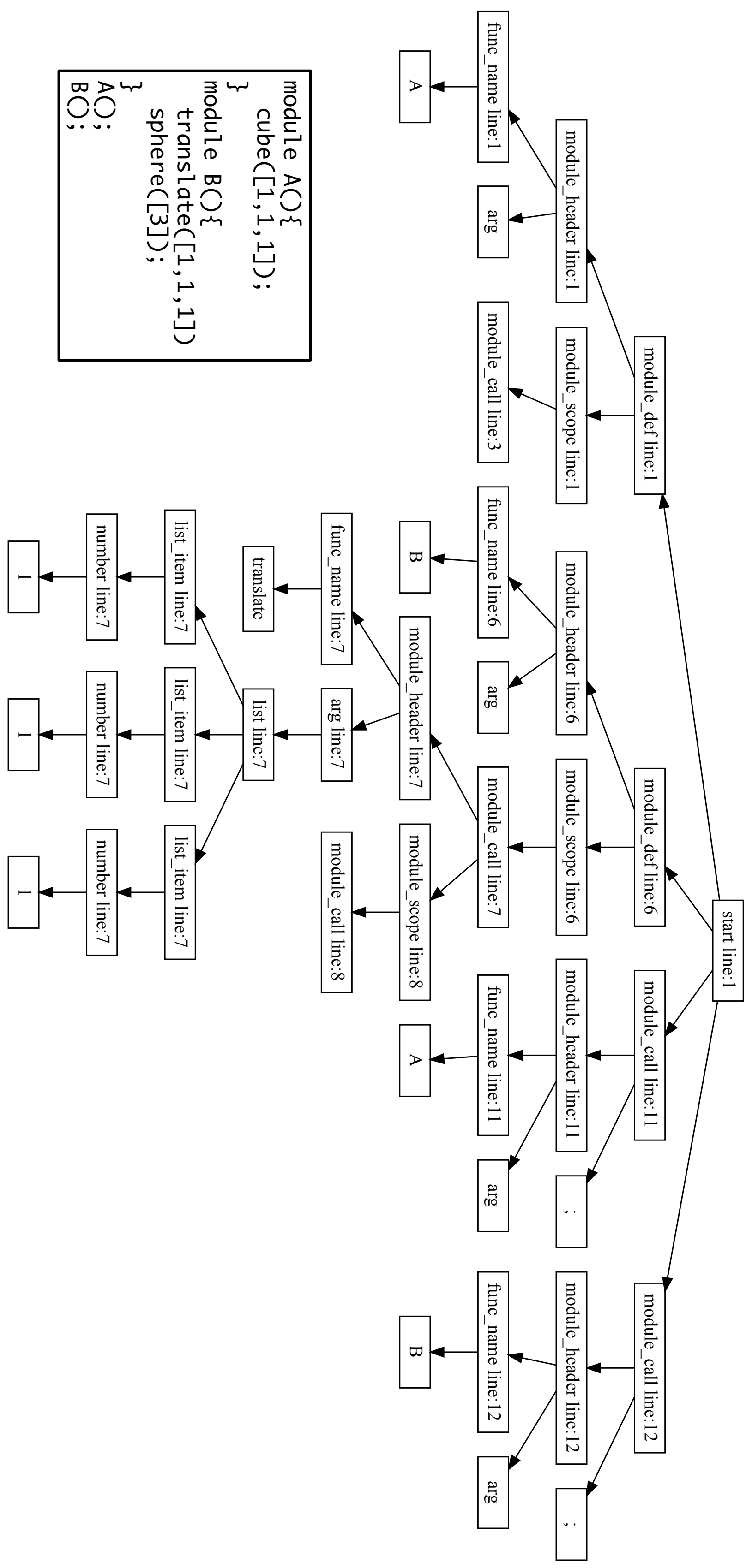


id: 6. label: wing

no.1

```
$vpf=10;
//gt label: body
color([0.832994562692686,0.00840958
translate([3.1180534698069096,-2.4010
rotate([89.64990470944257,
-0.12175487577762106,
-0.9554021632202875])
cube([30.387782454490665,4.29443761
//gt label: tail
color([1.0,0.5,0.0])
translate([-15.800148844718933,1.2916
rotate([177.9739741315627,
-1.502177810090334,
-55.15065451143329])
cube([8.262170851230621,3.90831075
//gt label: wing
color([0.01648220249221255,0.626268
translate([3.5263698175549507,-1.5767
rotate([-161.4573180345449,
-79.25417183996787,
57.45063059087617])
cube([7.19767458736897,6.55908968
//gt label: wing
color([0.0,0.0,1.0])
translate([1.3061624951660633,-0.8868
rotate([-70.67002271680099,
74.58483953896659,
14.431876011984228])
cube([10.466196089982988,2.2626117
//gt label: tail
color([0.5,1.0,0.5])
translate([-18.994220346212387,5.1878
rotate([-79.96940052911218,
89.36920978442605,
13.033560846167932])
cube([11.390546113252642,1.4712999
```

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



```
module A(){
  cube([1,1,1]);
}
module B(){
  translate([1,1,1])
    sphere([3]);
}
A();
B();
```

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