CAD-Assistant: Tool-Augmented VLLMs as Generic CAD Task Solvers

Dimitrios Mallis[1], Ahmet Serdar Karadeniz[1], Sebastian Cavada[1], Danila Rukhovich[1], Niki Foteinopoulou[1], Kseniya Cherenkova[2], Anis Kacem[1], Djamila Aouada[1]

[1]SnT, University of Luxembourg [2]Artec3D, Luxembourg

Project Page

https://cadassistant.github.io/

SCAN ME

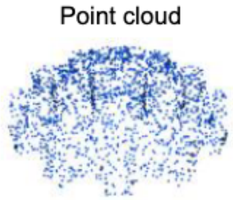
Github

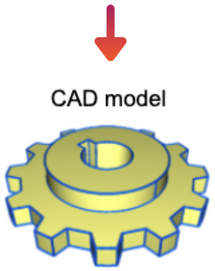
dimitrismallis/CAD-Assistant

SCAN ME

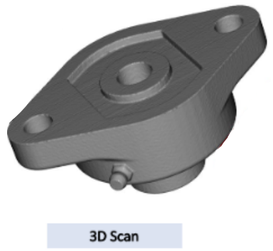
Introduction

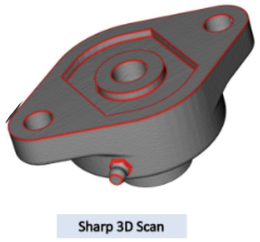
There is significant research interest in CAD automation but most research efforts focus on **fixed CAD workflows**.



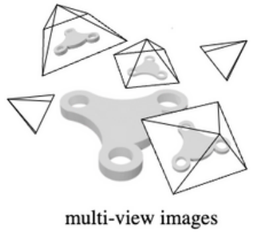


3D Reverse Engineering [1]





Edge Parameterization [2]



CAD from multiview images [3]

Hand-drawn sketch parametarization [4]

[1] Rukhovich, D. et al. “CAD-Recode: Reverse Engineering CAD Code from Point Clouds.” ICCV (2025).
[2] Cherenkova, K. et al. “SepicNet: Sharp Edges Recovery by Parametric Inference of Curves in 3D Shapes.” CVPRW (2023).
[3] Hong, E. et al. “MV2Cyl: Reconstructing 3D Extrusion Cylinders from Multi-View Images.” NeurIPS (2024).
[4] Karadeniz, A. et al. “DAVINCI: A Single-Stage Architecture for Constrained CAD Sketch Inference.” BMVC (2024).

Motivation

> Vision and Large Language Models (VLLMs) have potential for enabling AI-assisted CAD design.

Challenges

VLLMs have been shown to be weak in in **geometric reasoning** and **handling of mathematical concept**:

- Interpret CAD models from their CAD sequences [1].
- Recognize spatial arrangement [2].
- Correctly orient primitives [3].
- Predict the cumulative effects of the CAD commands.

[1] Qiu, Z. et al. “Can Large Language Models Understand Symbolic Graphics Programs?” ICLR (2025).
[2] Sharma, P. et al. “A Vision Check-up for Language Models.” CVPR (2024).
[3] Makatura, L. et al. “How Can Large Language Models Help Humans in Design And Manufacturing?” Arxiv (2024)

This work introduces CAD-Assistant

A generic tool-augmented VLLM framework that integrates CAD-specific tools to address the limitations of VLLMs in AI-assisted CAD.

`CAD-Assistant`: CAD-Specific Tools

CAD-Assistant is equipped with a diverse set of CAD-specific tools and can process multimodal inputs, including hand-drawn sketches and 3D scans.

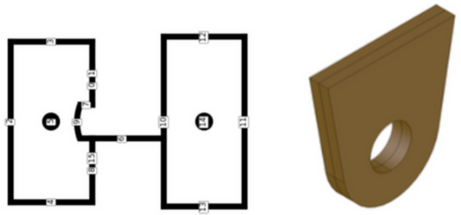
FreeCAD

CAD software intergration

Python

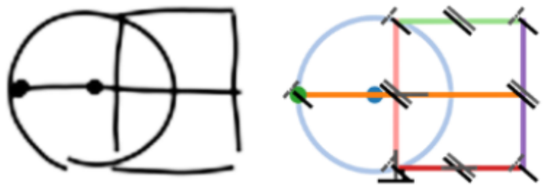
Actions and logical operations

CAD Recognizers



CAD sketch and CAD Model renderers

Sketch Parameterizer

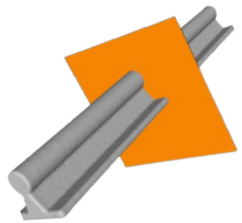


Hand-drawn image to constrained CAD sketch

Constraint Checker

Constraint analysis routine

Crosssection Extractor

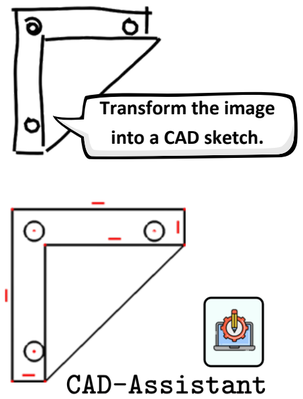


Extract cross-section from 3D mesh

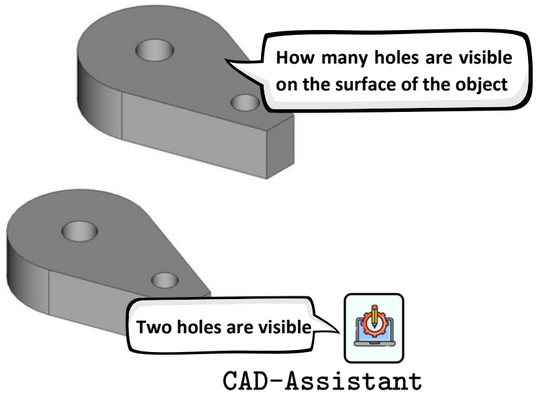
CAD-Assistant: Real-World Use Cases

We evaluate CAD-Assistant on existing benchmarks and demonstrate diverse real-world use cases.

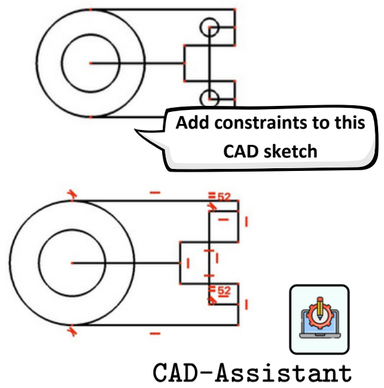
Handdrawn Image Parameterization



CAD Question Answering

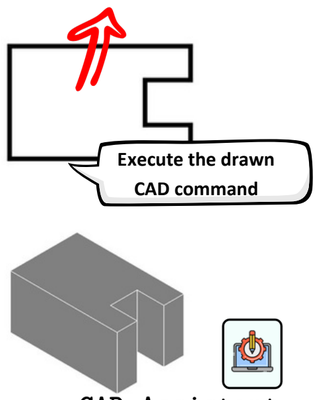


CAD Sketch Autoconstraining

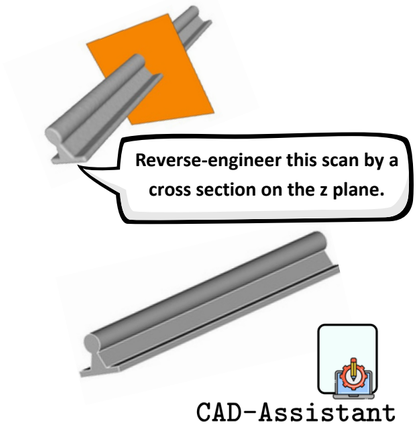


Handdrawn sketch to 3D solid

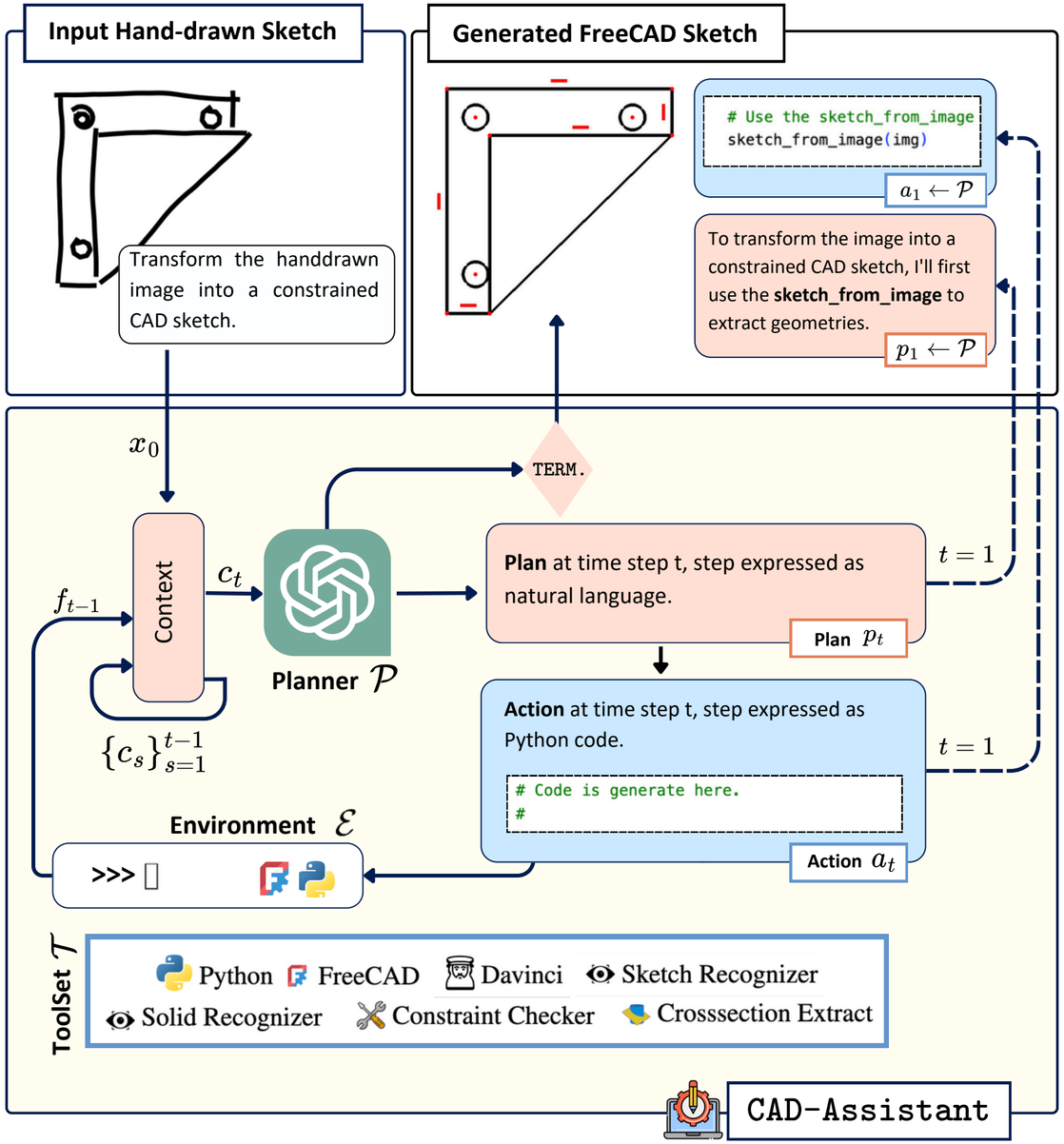
Visual Editing



3D Reverse Engineering

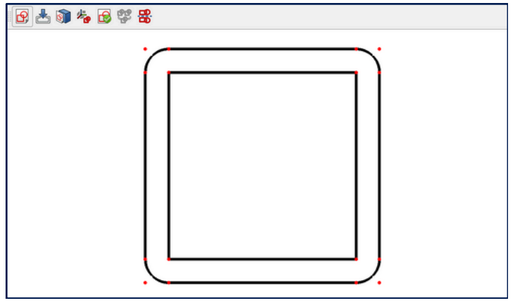


CAD-Assistant: Framework Overview



CAD-Assistant: Autoconstraining Trajectory

USER QUERY: Analyse the sketch to identify geometric relationships and apply parametric constraints.



PLAN: To analyze the sketch and apply the appropriate parametric constraints, I will first use the *sketch_recognizer* function to understand the current state of the sketch. Then I will use the analyse_constraints function to evaluate their validity and impact on the sketch before applying them.

ACTION: Use the *sketch_recognizer* function to analyze the current state of the sketch.

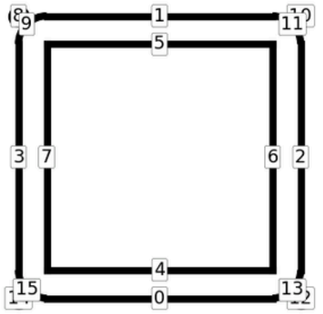
```
# Use the sketch_recognizer function to analyze the current state of the sketch
sketch_recognizer(sketch)
```

ENVIRONMENT: Execution Success

The sketch contains the following elements, serialized in JSON format:


```
{"Geometry":[{"Id":0,"Type":"Line segment","StartPoint":{"x":0.4,"y":-0.5},
"EndPoint":{"x":-0.4,"y":-0.5},"isConstruction":false},{"Id":1,"Type":
"Line segment","StartPoint":{"x":0.4,"y":0.5},"EndPoint":{"x":-0.4,"y":0.5}
,"isConstruction":false},{"Id":2,"Type":"Line segment","StartPoint":{"x":0.5,
"y":-0.4},"EndPoint":{"x":0.5,"y":0.4},"isConstruction":false},{"Id":3,"Type":
"Line segment","StartPoint":{"x":-0.5,"y":-0.4},"EndPoint":{"x":-0.5,"y":0.4},
"isConstruction":false},{"Id":4,"Type":"Line segment","StartPoint":{"x":0.4,"y"
:-0.4},"EndPoint":{"x":-0.4,"y":-0.4},"isConstruction":false},{"Id":5,"Type":
                                                              .
                                                              .
                                                              .
"Constraints":[]}
```


Rendered Sketch Image



CAD-Assistant: Autoconstraining Trajectory

PLAN: The sketch consists of several line segments and arcs forming a square with rounded corners. To ensure the sketch is fully constrained, I will apply the following constraints:

1. Coincident constraints to connect endpoints of adjacent geometries.

2. Horizontal and Vertical constraints for the straight line segments.

3. Equal constraints for the arcs to ensure they have the same radius.

4. Tangent constraints between the arcs and the line segments.

I will first analyse and apply the Coincident constraints.

ACTION: Analyze and apply Coincident constraints to connect geometries.

```
# Define Coincident constraints for connecting endpoints
coincident_constraints = [
    Sketcher.Constraint('Coincident', 0, SubRef.START.value, 15, SubRef.END.value),
    Sketcher.Constraint('Coincident', 0, SubRef.END.value, 13, SubRef.START.value),
    Sketcher.Constraint('Coincident', 1, SubRef.START.value, 9, SubRef.END.value),
    Sketcher.Constraint('Coincident', 1, SubRef.END.value, 11, SubRef.START.value),
    Sketcher.Constraint('Coincident', 2, SubRef.START.value, 11, SubRef.END.value),
    Sketcher.Constraint('Coincident', 2, SubRef.END.value, 13, SubRef.END.value),
    Sketcher.Constraint('Coincident', 3, SubRef.START.value, 15, SubRef.START.value),
    Sketcher.Constraint('Coincident', 3, SubRef.END.value, 9, SubRef.START.value),
    Sketcher.Constraint('Coincident', 4, SubRef.START.value, 7, SubRef.END.value),
    Sketcher.Constraint('Coincident', 4, SubRef.END.value, 6, SubRef.START.value),
    Sketcher.Constraint('Coincident', 5, SubRef.START.value, 7, SubRef.START.value),
    Sketcher.Constraint('Coincident', 5, SubRef.END.value, 6, SubRef.END.value)
]
# Analyze and apply constraints
for i, constraint in enumerate(coincident_constraints):
    is_valid, moves_geometries = analyse_constraint(sketch, constraint)
    if is_valid and not moves_geometries:
        sketch.addConstraint(constraint)
sketch.recompute()
```

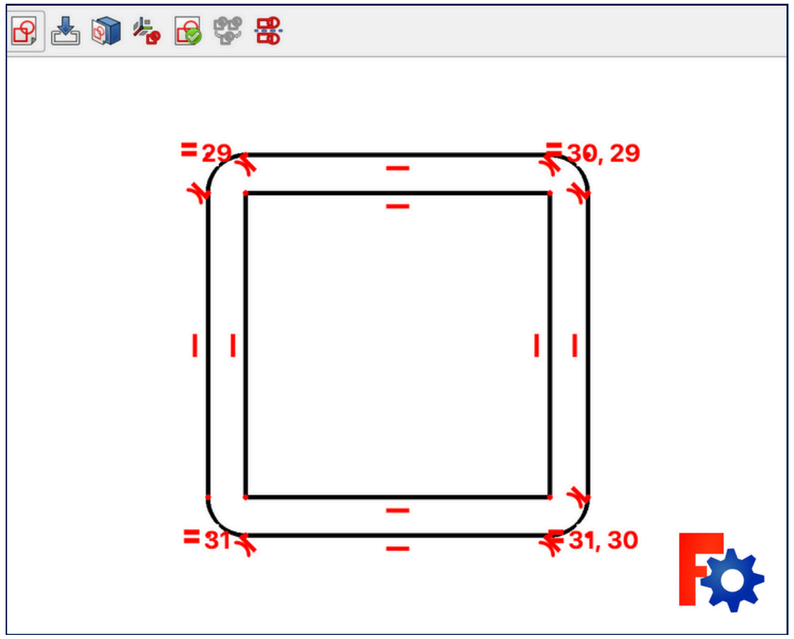
ENVIRONMENT: Execution Success

CAD-Assistant: Autoconstraining Trajectory

PLAN: The Coincidents have been applied, ensuring that the loops are closed. Next, I will apply Horizontal and Vertical constraints to the straight line segments to maintain their orientation.

(Rest of the steps)

PLAN: The sketch is now properly constrained. **TERMINATE**.



Tool Use for Effective Geometric Reasoning

VLLMs have shown **limited ability** to geometrically comprehend CAD programs. We examine **CAD representations** that can be derived via tool-use to improve geometric reasoning.

Parametrization Strategies

Implicit

```
Line = (
  refX, refY,
  dirX, dirY,
  startDist, endDist
)

Arc = (
  centerX, centerY,
  dirX, dirY,
  clockwise,
  startAngle, endAngle
)
```

Point-based

```
Line = (
  startX, startY,
  endX, endY
)

Arc = (
  startX, startY,
  midX, midY,
  endX, endY
)
```

Over-parameterized

```
Line = (
  refX, refY, dirX, dirY, startDist, endDist,
  startX, startY, endX, endY
)

Arc = (
  centerX, centerY, dirX, dirY, startX, startY,
  midX, midY, endX, endY, clockwise, startAngle, endAngle
)
```

Serialization Strategies

Json


```
{"Geometry":[{"Id":0,"Type":"Line segment","StartPoint":{"x":0.4,"y":-0.5},
"EndPoint":{"x":-0.4,"y":-0.5},"isConstruction":false},{"Id":1,"Type":
"Line segment","StartPoint":{"x":0.4,"y":0.5},"EndPoint":{"x":-0.4,"y":0.5}
,"isConstruction":false},{"Id":2,"Type":"Line segment","StartPoint":{"x":0.5,
"y":-0.4},"EndPoint":{"x":0.5,"y":0.4},"isConstruction":false},{"Id":3,"Type":
"Line segment","StartPoint":{"x":-0.5,"y":-0.4},"EndPoint":{"x":-0.5,"y":0.4},
"isConstruction":false},{"Id":4,"Type":"Line segment","StartPoint":{"x":0.4,"y"
:-0.4},"EndPoint":{"x":-0.4,"y":-0.4},"isConstruction":false},{"Id":5,"Type":
```

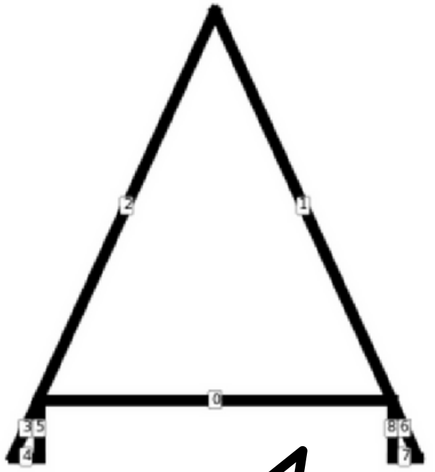

HTML


```
<table border="1" cellspacing="0" cellpadding="4">
  <tr>
    <th>Id</th><th>Type</th><th>StartPoint_x</th><th>StartPoint_y</th><th>EndPoint_x</th><th>EndPoint_y</th>
  </tr>
  <tr><td>0</td><td>Line segment</td><td>0.4</td><td>-0.5</td><td>-0.4</td><td>-0.5</td><td>false</td></tr>
  <tr><td>1</td><td>Line segment</td><td>0.4</td><td>0.5</td><td>-0.4</td><td>0.5</td><td>false</td></tr>
  <tr><td>2</td><td>Line segment</td><td>0.5</td><td>-0.4</td><td>0.5</td><td>0.4</td><td>false</td></tr>
  <tr><td>3</td><td>Line segment</td><td>-0.5</td><td>-0.4</td><td>-0.5</td><td>0.4</td><td>false</td></tr>
  <tr><td>4</td><td>Line segment</td><td>0.4</td><td>-0.4</td><td>-0.4</td><td>-0.4</td><td>false</td></tr>
  <tr><td>5</td><td>Line segment</td><td>0.4</td><td>0.4</td><td>-0.4</td><td>0.4</td><td>false</td></tr>
</table>
```


CSV


```
Id,Type,Start_x,Start_y,End_x,End_y,isConstr
0,"Line segment",0.4,-0.5,-0.4,-0.5,false
1,"Line segment",0.4,0.5,-0.4,0.5,false
2,"Line segment",0.5,-0.4,0.5,0.4,false
3,"Line segment",-0.5,-0.4,-0.5,0.4,false
4,"Line segment",0.4,-0.4,-0.4,-0.4,false
5,"Line segment",0.4,0.4,-0.4,0.4,false
```


Rendering-based Reasoning



What type of triangle is shown in the **image** based on its side lengths?
A) Equilateral B) Isosceles
C) Scalene D) Right

Tool Use for Effective Geometric Reasoning

Investigation of tool-derived CAD representations for 2D CAD QA.

Text-based Reasoning

Serialization	Parametarization	*2D Acc*
Common CAD Sketch formats		
Serialized Graph	Implicit	0.674
DXF		0.671
OCA		0.707
Serialization Strategy (Tabular formats)		
CSV	Point-based	0.703
Markdown	Point-based	0.706
HTML	Point-based	0.710
Serialization Strategy (Schema-embedded formats)		
Serialized Graph	Point-based	0.744
JSON	Point-based	**0.748**
JSON	Overparametarized	0.747

Rendering-based Reasoning

CAD Sketch Image Type	
Hand-drawn Sketch	0.616
Precise Rendering	**0.754**

Findings:

- Schema-embedded representation performs better than tabular formats.
- The Planner is sensitive to the geometry parameterization.
- Rendering-based question answering surpasses text-based recognition.

Tool Use for Effective Geometric Reasoning

The proposed zero-shot method is evaluated on standard CAD benchmarks. The CAD-Assistant **outperforms** baselines and task-specific approaches.

CAD Question Answering

Method	Planner	*2D Acc*	*3D Acc*
SGPBench [1]	GPT-4 mini	0.594	0.737
	GPT-4 Turbo	0.674	0.762
	GPT-4o	0.686	0.782
CAD-ASSISTANT	GPT-4 mini	0.614	0.783
	GPT-4 Turbo	0.741	0.825
	GPT-4o	**0.791**	**0.857**

CAD Sketch Autoconstraining

Method	Type	*PF1* ↑	*CF1* ↑
GPT-4o	*zero-shot*	0.693	0.274
Vitruvion [2]	*supervised*	0.706	0.238
CAD-ASSISTANT	*zero-shot*	**0.979**	**0.484**

Hand-drawn sketch image Parameterization

Method	*Acc* ↑	*CD* ↓
Vitruvion [2]	0.659	1.586
Davinci [3]	**0.789**	1.184
CAD-ASSISTANT	0.784	**0.680**

[1] Qiu, Z. et al. “Can Large Language Models Understand Symbolic Graphics Programs?” ICLR (2025).
[2] Seff et al. “Vitruvion: A Generative Model of Parametric CAD Sketches.” ICLR (2022).
[3] Karadeniz et al. “DAVINCI: A Single-Stage Architecture for Constrained CAD Sketch Inference.” BMVC (2024).

ICCV HONOLULU HAWAII
OCT 19-23, 2025

uni.lu | SnT

Project Page

https://cadassistant.github.io/

SCAN ME

Thank You!

Github

dimitrismallis/CAD-Assistant

SCAN ME

Artec 3D

Luxembourg National Research Fund