



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

Dimitrios Mallis¹

dimitrios.mallis@uni.lu

Ahmet Serda Karadeniz¹

ahmet.karadeniz@uni.lu

Sebastian Cavada¹

sebastian.cavada.dev@gmail.com

Danila Rukhovich¹

danila.rukhovich@uni.lu

Niki Foteinopoulou¹

niki.foteinopoulou@uni.lu

Kseniya Cherenkova^{1,2}

kseniya.cherenkova@uni.lu

Anis Kacem¹

anis.kacem@uni.lu

Djamila Aouada¹

djamila.aouada@uni.lu

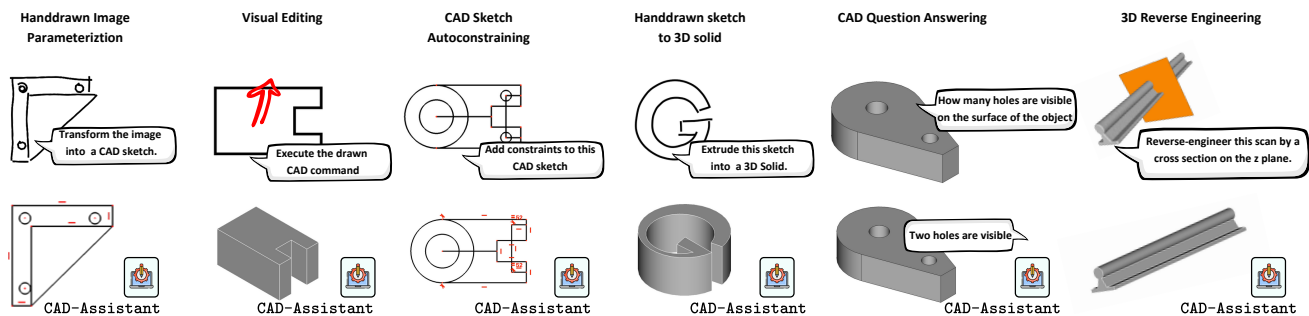
¹SnT, University of Luxembourg ²Artec3D, Luxembourg

Figure 1. CAD-Assistant is a tool-augmented VLLM framework for AI-assisted CAD. Our framework generated FreeCAD [11] code that is executed within CAD software directly and can process multimodal inputs, including textual queries, sketches, drawn commands and 3D scans. This figure showcases various examples of generic CAD queries and the responses generated by CAD-Assistant.

Abstract

We propose CAD-Assistant, a general-purpose CAD agent for AI-assisted design. Our approach is based on a powerful Vision and Large Language Model (VLLM) as a planner and a tool-augmentation paradigm using CAD-specific tools. CAD-Assistant addresses multimodal user queries by generating actions that are iteratively executed on a Python interpreter equipped with the FreeCAD [11] software, accessed via its Python API. Our framework is able to assess the impact of generated CAD commands on geometry and adapts subsequent actions based on the evolving state of the CAD design. We consider a wide range of CAD-specific tools including a sketch image parameterizer [21], rendering modules, a 2D cross-section generator, and other specialized routines. CAD-Assistant is evaluated on multiple CAD benchmarks, where it outperforms VLLM baselines and supervised task-specific methods. Beyond existing benchmarks, we qualitatively demonstrate the potential of tool-augmented VLLMs as general-purpose CAD

solvers across diverse workflows. Code implementation of the CAD-Assistant framework is publicly available <https://github.com/dimitrismallis/CAD-Assistant>.

1. Introduction

Computer-Aided Design (CAD) refers to the use of computer software to assist in the creation, modification, analysis, or optimization of a design [5]. Recently, there has been a significant research interest in the automation of CAD pipelines. Examples include, 3D reverse-engineering [13, 23, 37, 55], CAD generation [47, 49, 58, 61], edge parametrization [9, 72], CAD from multiview images [17, 68], hand-drawn CAD sketch parametrization [21, 22] and text-guided CAD editing [26]. Still, most efforts to date have centered around fixed workflows, and the development of CAD agents to address generic tasks remains largely unexplored. In this work, we advocate that the creation of CAD agents capable of interacting with and

supporting designers through the CAD process, would be a transformative advancement for the CAD industry.

As Vision and Large Language Models (VLLMs) continue to mature [1, 3, 12, 28, 31, 32, 38, 40], they hold promise for enabling AI-assisted CAD design, particularly given that their very vast pre-training endow them with broad knowledge of design and manufacturing [36]. Despite the identified potential, their ability to be used within computational design and manufacturing workflows remains severely constrained by weaknesses in geometric reasoning and handling of mathematical concepts [19]. Indeed, VLLMs may struggle to semantically interpret the appearance of rendered objects from their corresponding CAD sequences [45]. They may also fail to recognize spatial arrangements and the varied combinations of visual concepts [50] or correctly orient primitives and generate accurate placements [36]. Their effectiveness in an agentic CAD setting is further hindered by the inherently unpredictable effects of CAD commands. High-level CAD operations, such as applying geometric constraints, fillet, chamfer, etc, can have complex and non-intuitive impacts on a model's geometry and topology [48, 49], which is typically resolved by advanced CAD solvers. VLLMs cannot reliably predict the cumulative effects of the CAD commands they generate further limiting their practical usability in CAD workflows.

Recently, tool-augmentation has emerged as a prevailing strategy for addressing various shortcomings of foundational models and enhancing their performance in real-world applications [19, 34, 51, 54, 56]. Despite demonstrated effectiveness, VLLMs capable of composing and utilizing external tools have yet to be explored within the domain of CAD design. This work addresses this gap by introducing *CAD-Assistant*, a generic tool-augmented VLLM framework that integrates CAD-specific tools to effectively address the limitations of VLLMs in AI-assisted CAD. *CAD-Assistant* integrates a wide range of external CAD-specific modules, including a hand-drawn image parameterizer, rendering modules for multimodal CAD sequence understanding, a specialized utility for analysis of geometric constraints and a 2D cross-section generator for VLLM interaction with 3D scans.

Our framework leverages a VLLM-based planner and CAD-specific tool augmentation for generic CAD task solving. The planner generates CAD code actions, that are executed directly within the open-source CAD software FreeCAD [11], accessed via its Python API. Geometric reasoning is enhanced by dedicated CAD rendering and parameter serialization modules, enabling a more comprehensive multimodal representation of CAD models throughout the planning and reasoning process. Instead of solely relying on the effect prediction of complex CAD commands, our CAD agent inspects the evolving state of a design and refines or corrects actions based on the current CAD geometry. CAD-

specific tools facilitate the processing of multimodal inputs, from text to hand-drawn sketches, precise CAD drawings, drawn commands and 3D scans.

CAD-Assistant is a *training-free* framework that generates CAD code on an open-source CAD API, producing outputs that are both editable and highly interpretable. *CAD-Assistant* is also highly extensible and can operate across the diverse set of commands available in the FreeCAD API, requiring only a Python docstring to incorporate further capabilities. This is in contrast to the majority of CAD automation research focusing on the limited set of CAD operations captured in large-scale CAD datasets [29, 58, 62, 63]. To address the lack of benchmarks for tool use akin to specialized sets commonly used in other domains [33, 35], this work adopts an evaluation setting for generic CAD agents leveraging multiple existing CAD tasks. Evaluations are conducted for 2D and 3D CAD question answering, auto-constraining, and hand-drawn CAD sketch image parametrization. *CAD-Assistant* outperforms both VLLM baselines and supervised task-specific methods trained on large-scale datasets, despite being prompted in a zero-shot manner. Furthermore, we demonstrate the potential of *CAD-Assistant* beyond existing benchmarks by showcasing diverse use cases, including generating 3D solids from hand-drawn sketches, performing 3D reverse engineering from 3D scans via cross-section parameterization, and visual CAD design through semantically interpretable drawing commands (*e.g.* sketching an extrusion operation). Example responses of the proposed *CAD-Assistant* on diverse multimodal queries are depicted in Figure 1.

Contributions: The main contributions of this work can be summarized as follows:

1. We introduce *CAD-Assistant*, the first tool-augmented VLLM framework for generic CAD task solving. Our framework is equipped with a diverse set of CAD-specific tools and can process multimodal inputs, including hand-drawn sketches and 3D scans.
2. We demonstrate the effectiveness of tool-use for mitigating VLLMs' limitations on AI-assisted CAD. Geometric reasoning is enhanced by incorporating comprehensive multimodal representations of CAD models and enabling direct interaction with CAD software.
3. We propose a highly extensible and training-free framework that can operate beyond the simple set of CAD commands captured on existing CAD datasets.
4. We identify an evaluation setting for generic CAD agents based on existing benchmarks. The proposed zero-shot method outperforms baselines and task-specific approaches trained on large datasets. We also qualitatively demonstrate the potential of *CAD-Assistant* on a diverse set of real-world use cases.

2. Related Work

Foundation Models for CAD: Recently, there has been increasing research interest in the use of foundation models on CAD-related applications. CAD-Talk [69] introduces a framework for semantic CAD code captioning using multi-view photorealistic renderings of CAD models along with part-segmentation, powered by foundation models [8, 25]. Taking a similar path, QueryCAD [24] proposes an open-vocabulary CAD part segmentation from images leveraging segmentation foundation models and LLMs to perform CAD-related question-answering for robotic applications. CADLLM [60] proposes a T5 model [46] finetuned on the SketchGraphs [48] dataset of 2D CAD sketches for sketch auto-completion. CadVLM [59] extends CADLLM [60] to the visual domain, incorporating a visual modality for CAD sketch auto-completion, autoconstraining and image-guided generation. CADReparam [26] uses VLLMs to infer meaningful variation spaces for parametric CAD models, re-parameterizing them to enable exploration along design-relevant axes. Img2CAD [68] utilizes a VLLM to reverse engineer objects from images, predicting the specific CAD command types needed to model each part of the object accurately. Badagabettu *et al.* [4], focus on text-guided generation of CAD models as CADQuery code, while LLM4CAD [30] use a similar approach to generate 3D CAD models from text and image inputs. Related to ours is the training-free method of [2] focusing on CAD model generation. Authors introduce a verification process to ensure the validity of generated models, but do not explore tool augmentation. Our investigation diverges from these task-specific approaches as it shifts the focus on tool-augmentation for mitigating the limitation of VLLMs on AI-assisted CAD. CAD-Assistant is the first *general-purpose* framework for CAD design, able to process multimodal prompts and address diverse CAD use cases.

Tool-augmented VLLMs: Recently there has been growing interest in enhancing LLMs and VLLM performance via augmentation with external tools [16, 19, 34, 51, 54, 56, 66, 70]. The field is further propelled by the emergence of benchmarks, namely ScienceQA [33] and TabMWP [35], which are well-suited for evaluating the effectiveness of tool-use. Tool-use offers several benefits [44], such as reducing hallucinated knowledge [52], providing real-time information [34], enhancing domain expertise [39] and producing interpretable outputs by making intermediate steps explicit [16, 54]. Planning is commonly performed via instructions in natural language [16, 34] or Python code generation [19, 54], and tool set might include search engines [27, 34, 39], calculators [10, 42], external APIs [43], vision modules [16, 54], Hugging Face models [51], Azure models [66] or LLM created tools [7]. Despite the vast potential of tool-augmented LLMs and VLLMs for CAD-

related applications, the space remains unexplored. To our knowledge, this work is the first investigation on tool-augmented VLLMs for AI-assisted CAD.

VLLMs as Geometrical Reasoners: In order to advance tool-augmented VLLMs for AI-assisted CAD, it is crucial for the VLLMs planner to semantically recognize and precisely identify and manipulate individual elements within parametric geometries. This type of precision is an essential skill when interfacing with CAD software. Naturally, this raises the question: *Can large vision language models understand symbolic graphics programs?* In that direction, Yi *et al.* [67] explored incorporating symbolic structure as prior knowledge for enhancing visual question answering. More recently, Sharma *et al.* [50] examined visual program generation and recognition, showing that while shape generation often relies on memorizing prototypes from training data, shape recognition demands a deeper understanding of primitives. Qi *et al.* [45] introduced SGPBench, a question-answering benchmark designed to assess the semantic understanding and consistency of symbolic graphics programs, including CAD models. This benchmark evaluates the extent of LLMs’ ability to semantically comprehend and reason about geometric structures. While [45] applied instruction tuning to improve visual program understanding, our work emphasizes general-purpose VLLMs, demonstrating that factors like serialization and parametrization strategies for formatting geometry and multimodal representation of a CAD model can significantly expand VLLMs’ capacity for geometric reasoning.

3. The proposed CAD-ASSISTANT

3.1. General Framework

This section provides an overview of CAD-ASSISTANT. Our framework comprises the following three components:

Planner: The planner \mathcal{P} is modelled by a VLLM capable of advanced reasoning. Following [19], on each timestep t , the planner analyses the current context c_t and generates a plan p_t and an action a_t that implements p_t . In this work, we employ GPT-4o [40] as the core framework planner.

Environment: We utilize the Python interpreter as the primary environment \mathcal{E} for executing the generated action a_t at time t . Additionally, \mathcal{E} integrates CAD software [11] as a foundational component for AI-assisted CAD applications. On each timestep, t , the environment provides feedback e_t of the current state of the CAD design.

Tool Set: CAD-ASSISTANT utilizes a set $\mathcal{T} = \{\mathcal{T}_i\}_{i=1}^N$ of N CAD-specific tools, suitable for AI-Assisted CAD. These include standard Python libraries, modules of the FreeCAD Python API [11] to interface CAD commands, and other useful CAD-specific tools and Python routines. CAD-ASSISTANT can be formalized as follows: Given

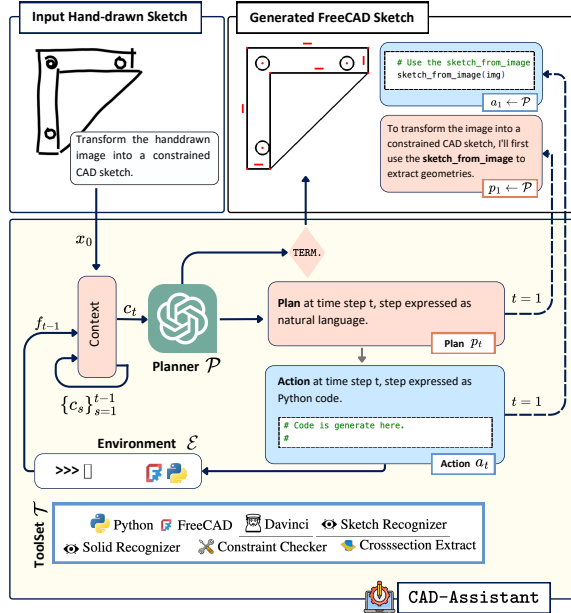


Figure 2. Overview of CAD-Assistant framework. A multi-modal user request is provided as context to a VLLM planner \mathcal{P} . At step t , the planner generates a plan p_t and an action a_t (python code). The action is executed on an environment \mathcal{E} and the generated execution output f_t is fed back to the planner, enabling generation for the next timestep.

a multimodal x_0 input user query, on each timestep t , the planner \mathcal{P} generates:

$$p_t \leftarrow \mathcal{P}(x_0; c_{t-1}, \mathcal{T}), \quad (1)$$

$$a_t \leftarrow \mathcal{P}(p_t; c_{t-1}, x_0, \mathcal{T}), \quad (2)$$

where p_t is the current plan in natural language, and a_t is the current action formulated as Python code. Then, the generated action a_t is executed on the framework’s environment:

$$(f_t, e_t) \leftarrow \mathcal{E}(a_t; e_{t-1}, \mathcal{T}, x_0), \quad (3)$$

where f_t is the output of the code execution, and e_t is the new state of the CAD design. Note that f_t can include both textual and visual outputs of the execution, e.g. list of CAD geometries in `.json` format or the rendering of the current state of the CAD object. Finally, the context is updated as:

$$c_{t+1} \leftarrow \text{concat}(f_t, \{c_s\}_{s=1}^t), \quad (4)$$

concatenating the previous context with the current code execution output and is supplied to \mathcal{P} for plan generation of timestep $t+1$. This process iterates for an arbitrary T number of steps until the planner \mathcal{P} concludes that the request x_0 has been successfully addressed. At that point, \mathcal{P} generates p_T , a special TERMINATE plan that indicates the completion of CAD-ASSISTANT’s response. An illustration of the proposed execution flow is provided in Figure 2 and an example of the agents’ trajectory as a response to an auto-constraining request is provided in Figure 3.

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": 0.5,
    "y": 0.5}, "isConstruction": false}, {"id": 2, "Type": "Line segment",
    "StartPoint": {"x": 0.5, "y": 0.5}, "EndPoint": {"x": 0.5, "y": 0.4},
    "isConstruction": false}, {"id": 3, "Type": "Line segment", "StartPoint":
    {"x": 0.5, "y": 0.4}, "EndPoint": {"x": 0.4, "y": 0.4}, "isConstruction":
    false}, {"id": 4, "Type": "Line segment", "StartPoint": {"x": 0.4, "y": 0.4},
    "EndPoint": {"x": 0.4, "y": 0.5}, "isConstruction": false}, {"id": 5, "Type":
    "Arc", "StartPoint": {"x": 0.4, "y": 0.4}, "EndPoint": {"x": 0.5, "y": 0.4},
    "isConstruction": false}, {"id": 6, "Type": "Arc", "StartPoint": {"x": 0.5,
    "y": 0.4}, "EndPoint": {"x": 0.5, "y": 0.5}, "isConstruction": false},
    "id": 7, "Type": "Arc", "StartPoint": {"x": 0.5, "y": 0.5}, "EndPoint":
    {"x": 0.4, "y": 0.5}, "isConstruction": false}, {"id": 8, "Type": "Arc",
    "StartPoint": {"x": 0.4, "y": 0.5}, "EndPoint": {"x": 0.4, "y": 0.4},
    "isConstruction": false}],
  "Constraints": []
}
```

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 endpoints of 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.START.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

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.

ACTION: Apply Horizontal and Vertical constraints to the line segments.

```
# Define Horizontal and Vertical constraints
# Rest of the code here.
```

PLAN: The sketch is now properly constrained.
TERMINATE:

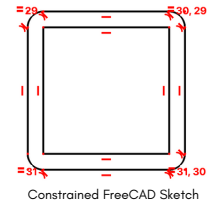


Figure 3. Execution flow for autoconstraining. The sketch recognizer function is utilized for multimodal CAD understanding. Constraints are generated over multiple timesteps.

3.2. CAD-specific Tool-set

CAD-ASSISTANT includes a set of N CAD-specific tools or modules. Each tool is defined by its method signature and the docstring [18] that disambiguates its use. Mod-








Module Type	Module Description
 Python	Action format and logical operations.
 FreeCAD	Integration with CAD software [11].
 Sketch Parameterizer	Hand-drawn sketch image to CAD sketch based via [21].
 Sketch Recognizer	Renders sketch and plots parameters.
 Solid Recognizer	Renders a 3D CAD model and plots parameters.
 Constraint Checker	Analyzes geometric constraints.
 Crosssection Extract	Generates an image of a cross section from a 3D mesh.

Table 1. Overview of CAD-specific tools.

ules \mathcal{T}_i are instantiated via their Python interface with arguments generated by \mathcal{P} as part of the action a_t . Notably, actions are formulated as Python code, as in [19, 54], rather than the natural-language instructions advocated by recent works [16, 34]. This design choice allows for direct use of the FreeCAD API. Moreover, the generated action a_t can access the parameters of the CAD models’ state e_t and perform logical and computational operations, which is highly advantageous for design tasks (see also section 10 of supplementary). Our CAD-specific tool set is summarized in Table 1 and a detailed overview of each tool is provided in supplementary.

4. Experiments

This section outlines the experiments conducted to validate the effectiveness of CAD-Assistant.

4.1. Strategies for Effective Geometric Reasoning

Effective geometric reasoning is an essential requirement for the development of generic CAD agents. However, VLLMs have shown limited ability to geometrically comprehend and mathematically reason about CAD programs [45, 50]. Previous work has explored symbolic instruction tuning [45] for addressing this limitation. In contrast, we shift our focus on tool augmentation as a training-free alternative to enhance geometric reasoning. This subsection examines CAD representations that can be derived using external tools and improve VLLMs’ understanding of CAD programs. Specifically, we study the following factors of a CAD representation:

Parametrization Strategy: Parametric geometries can be represented by different sets of parameters. For instance, a line could use start / end points or an angle and length relative to a reference. We compare the implicit parametrization approach of [45] to the point-based primitive representation of [22]. We also explore over-parametrization, where a redundant set of parameters is used per geometry. More de-

2D CAD SGPBench - Sketch in Textual Format

Serialization	Parameterization	Acc
<i>SGPBench [45] format</i>		
Serialized Graph	Implicit	0.674
<i>Standardized CAD Sketch formats</i>		
DXF [20]		0.671
OCA [15]		0.707
<i>Serialization Strategy (Tabular formats)</i>		
CSV	Point-based	0.703
Markdown	Point-based	0.706
HTML	Point-based	0.710
<i>Serialization Strategy (Schema-embedded formats)</i>		
Serialized Graph	Point-based	0.744
JSON	Point-based	0.748
<i>Parameterization Strategy</i>		
JSON	Point-based	0.748
JSON	Overparameterized	0.747

2D CAD SGPBench - Sketch as a Rendering

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

Table 2. Investigation of prompting strategies on geometric reasoning. We report performance for GPT-4o in terms of accuracy on the 2D partition of SGPBench [45]. **(Top)** Impact of Parameterization and serialization on CQA performance. **(Bottom)** Performance from hand-drawn and precise rendering of a CAD sketch.

tails about this comparison are provided on supplementary.

Serialization Strategy: The serialization format used to convert the parametric geometry into text can impact the planner’s ability to understand the geometry. Motivated by recent work on text-based serialization methods for tabular data [14], we compare commonly used formats such as *CSV*, *Markdown*, *HTML*, and *JSON*.

Rendering-based Reasoning: We investigate visual representations for geometric reasoning by providing the VLLM planner with 2D renderings of the CAD sketch or 3D solid.

To examine the impact of the above strategies on CAD program understanding and geometric reasoning, we experiment on the CAD question answering benchmark SGPBench [45]. This benchmark comprises multiple-choice questions and captures three types of graphical programs, *i.e.*, SVG, CAD sketches, and 3D CAD models. For this experiment, we report accuracy on the 2D CAD subset. This subset is derived from 700 CAD sketches from Sketch-Graphs [48]. A VLLM planner (GPT-4o) is provided with a textual description of a 2D CAD sketch and tasked with

Method	Planner	2D Acc	3D Acc
SGPBench [45]	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

Table 3. Comparison for the proposed CAD-ASSISTANT to baselines for CQA on the 2D and 3D subsets of SGPBench [45]. For CAD-ASSISTANT performance is reported for different planners.

answering a multiple-choice question about the design.

In Table 2 (top), we analyze the effect on the performance of the parametrization and serialization strategies used to parse the CAD sketch into a textual format. Firstly, we observe that schema-embedded representation like *JSON* performs better than tabular formats. Note that this is in contrast with recent work [53], where *HTML* was identified as the optimal serialization for tabular data. Secondly, GPT-4o demonstrates high sensitivity to geometry parametrization. The implicit parametrization used in SGPBench [45] significantly under-performs compared to a point-based parametrization for geometric primitives as in [22]. Overall, using a *JSON* serialization along with the point-based parametrization from [22] leads to substantial improvements over the original SGPBench format and other text-based CAD sketch formats, such as *DXF* and *OCA*. While over-parameterizing the sketches results in a negligible drop in performance w.r.t. a point-based parameterization, we argue that it is safer to opt for over-parameterization as other tasks might benefit from it. Furthermore, as shown in Table 2 (bottom), rendering-based question answering surpasses the performance reported for text-based recognition. Following these findings, we equip the CAD-ASSISTANT with a specialized recognition tools that generate an over-parameterized *JSON* representation of CAD models as well as renderings of 2D CAD sketch or 3D solid for comprehensive multimodal geometric reasoning.

4.2. CAD Benchmarks and Experimental Setup

As a generic framework, CAD-ASSISTANT can be conditioned to perform a wide range of tasks related to CAD design. Given the lack of specialized evaluation benchmarks for CAD agents, this work adapts an evaluation setting based on the following existing CAD tasks.

CAD Question Answering: As in subsection 4.1, quantitative evaluations of CAD Question Answering (CQA) is performed on the recently introduced SGPBench [45]. We do not provide the CAD code as part of the prompt as in [45]. Instead, the CAD sketch or model is pre-loaded into a FreeCAD project file, allowing CAD-ASSISTANT to utilize the FreeCAD integration and CAD-specific tools

to understand the design and answer questions. This experimental setup simulates a real-world question-answering environment where a CAD designer can ask open-ended questions about the design to support the iterative design process. We report accuracy on the 2D and 3D CAD sets.

Autoconstraining: Parametric constraints are a key component of feature-based CAD modeling [37] and a widely adapted mechanism for explicit capturing of design intent [41, 71]. Given a CAD sketch of n parametric primitives $\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\} \in \mathcal{P}^n$ (lines, arcs, circles, points) the goal of autoconstraining is to infer a set of parametric constraints $\{\mathbf{c}_i\}_{i=1}^m \in \mathcal{C}^m$ applied on these primitives. Each constraint \mathbf{c}_i is composed of constraint type, participating primitives $\mathbf{p}_i, \mathbf{p}_j$ and subreferences (s_i, s_j) specifying the point of application (e.g. start, end, center). In contrast to the evaluation setting of [48, 49], we incorporate the application of the geometric solver (CAD software) to determine the final configuration of sketch primitives. Performance is measured in terms of Primitive F1 Score (*PFI*) and Constraint F1 Score (*CFI*) as in [65]. *PFI* defines a true positive as a primitive with the correct type and parameters within five quantization units, and for *CFI* a constraint is considered a true positive only if all associated primitives are also correctly predicted. Quantitative evaluations are performed on SketchGraphs [48]. We use the test set of [49] and evaluate on a subset of 700 CAD sketches due to the resource intensive nature GPT4-o API requests.

Hand-drawn CAD sketch Parameterization: Given a binary sketch image $\mathbf{X} \in \{0, 1\}^{h \times w}$, sketch parameterization aims to recover the complete constrained CAD sketch $(\{\mathbf{p}_i\}_{i=1}^n, \{\mathbf{c}_i\}_{i=1}^m)$. We report parametric accuracy computed on quantized primitive tokens as in [22, 49] after solving the CAD sketch. We also compute bidirectional Chamfer Distance (CD) on the image space. For evaluation, we use the same test split as in the autoconstraining task. For hand-drawn sketch synthesis, we follow the strategy of [49].

4.3. Experimental Results

We evaluate the performance of CAD-ASSISTANT on the benchmarks described in the previous section.

CAD Question Answering: CAD-ASSISTANT is able to interact directly with a CAD model via its integration with CAD software and is tasked with answering a question about the design. Results for CQA on SGPBench [45] are reported in Table 3. For this experiment, we also report the performance of the GPT-4 mini and GPT-4 Turbo models as planners. We observe that by leveraging available tools such as the Python interpreter and the comprehensive multimodal representation of CAD models generated via the recognizer tools, CAD-ASSISTANT improves CQA performance for both CAD sketches and 3D CAD models, thus highlighting the potential of tool-use for CAD understand-

Method	Type	$PF1 \uparrow$	$CF1 \uparrow$
GPT-4o	<i>zero-shot</i>	0.693	0.274
Vitruvion [49]	<i>supervised</i>	0.706	0.238
CAD-ASSISTANT	<i>zero-shot</i>	0.979	0.484

Table 4. Evaluation on the task of autoconstraining. Performance is measured in terms of $PF1$ and $CF1$ on the SketchGraphs [48].

CAD-Specific Tools		Prompting		$PF1 \uparrow$	$CF1 \uparrow$
<i>MMrecog</i>	<i>ConstrCheck</i>	<i>Demonstr</i>	<i>Docstr</i>		
×	×	<i>0-shot</i>	✓	0.726	0.318
✓	×	<i>0-shot</i>	✓	0.747	0.329
✓	✓	<i>0-shot</i>	✓	0.979	0.484
✓	✓	<i>5-shot</i>	×	0.981	0.514
✓	✓	<i>5-shot</i>	✓	0.984	0.529

Table 5. Impact of CAD-specific tools and prompting strategies for CAD-Assistant on the autoconstraining task.

ing. Notably, for the smaller GPT-4 mini, the performance gain from CAD-Assistant is marginally above (2D subset) or on-par (3D subset), emphasizing the need for pairing tool-augmented frameworks with a powerful VLLM.

Autoconstraining: We evaluate our method on the task of CAD sketch autoconstraining [49]. CAD-Assistant is prompted to apply a set of parametric constraints with proper design intent to a CAD sketch preloaded into a FreeCAD project file, similar to the CQA setup. Performance is compared to a GPT-4o baseline and the constraint generation model Vitruvion [49], trained on a large-scale dataset [48]. Results are reported on Table 4. Note that the autoconstraining performance is reported after solving the predicted constraints with a CAD solver. As we are operating within CAD software, the CAD solver enforces the predicted constraints (*e.g.*, orthogonality between two lines) on CAD sketches, adjusting the parameters of the affected primitives accordingly (*e.g.*, modifying the parameters of the two lines). We observe that both the baseline and [49] tend to generate poorly parameterized constraints, which may lead to the arbitrary repositioning of primitives when applied by the CAD solver, as evidenced by the low $PF1$ values. In contrast, CAD-Assistant effectively utilizes tools to interact with the CAD software, assesses the impact of constraints, and preserve the integrity of the geometry. Notably, constraints generated by CAD-Assistant result in a high $CF1$ score despite *zero-shot* prompting, further underscores the broad understanding of CAD-Assistant in CAD design. In Table 5, we investigate the impact of tools relevant to auto-constraining on the effectiveness of CAD-Assistant. We find that both the multimodal sketch recognizer (*MMrecog*) and the constraint checker module (*ConstrCheck*) contribute to performance gains. Table 5 also compares prompting strategies for the proposed framework. While we primary focus

Method	$Acc \uparrow$	$CD \downarrow$
Vitruvion [49]	0.659	1.586
Davinci [21]	0.789	1.184
CAD-ASSISTANT	0.784	0.680

Table 6. Evaluation on the task of hand-drawn image parametrization. Comparison against the task-specific models of [21, 49].

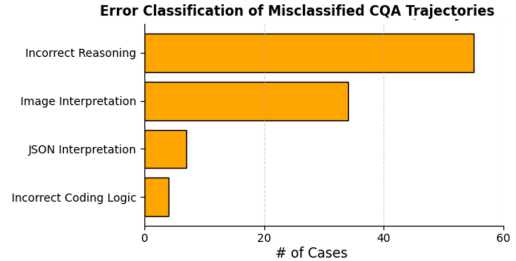


Figure 4. Classification of failure case types for erroneous responses in the CAD Question Answering task.

on *zero-shot* prompting, which promotes agentic behavior by eliminating the need for CAD designers to create tailored examples for unique use cases, we find that a few high quality demonstrations can further enhance performance as shown by the results for *5-shot* prompting.

Hand-drawn CAD sketch Parameterization: Our framework utilizes the sketch parameterization tool that processes hand-drawn inputs to generate a textual description of primitives and constraints, as well as the constraint analysis module to assess the impact of constraints on CAD geometry. Performance is compared to task-specific models in Table 6. We observe that CAD-Assistant effectively translates the text-based parameterization recovered by the sketch parameterizer (based on [21]) into a FreeCAD sketch, resulting in high accuracy. Additionally, it successfully applies constraints without compromising the solved geometry, as evidenced by the reduction in CD.

Human Evaluations. We conduct a failure case analysis on the CQA task, shown in Figure 4. A human annotator reviews 100 agent trajectories associated with misclassified answers and categorizes the error types. Most failures are caused from incorrect reasoning by the VLLM or misinterpretation of tool-generated renderings, such as visually confusing a trapezoid with a triangle. To evaluate the effectiveness of CAD-Assistant in tool usage, two human annotators examine 200 autoconstraining / parameterization trajectories for tool-use validity. The analysis shows a high validity rate of 98.5%, with the few errors observed primarily due to incorrect use of the FreeCAD API.

4.4. Exploring New Capabilities

Beyond Simplified CAD Commands: Research on common CAD tasks generally focuses on the limited sets of CAD commands captured by large-scale datasets [48, 61].

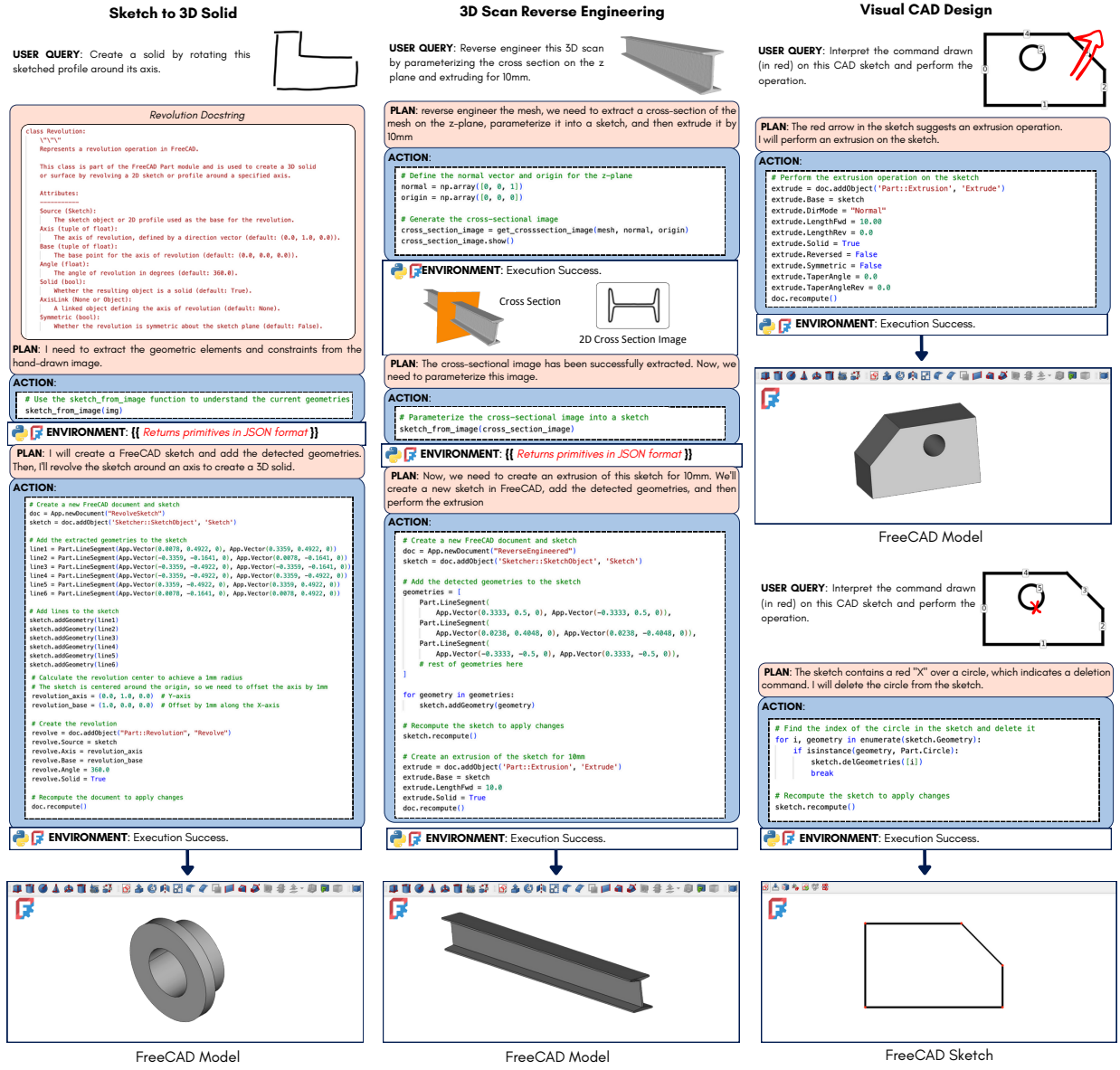


Figure 5. Real-world CAD use cases. (Left) The CAD-Assistant generated a 3D solid conditioned on a hand-drawn sketch image. (Center) Our method reconstructs a 3D scan via cross-section parameterization. (Right) The CAD-Assistant semantically interprets the drawn operation and fulfills user requests directly without composing CAD-specific tools.

As a train-free framework, CAD-Assistant can leverage the full range of commands available within the FreeCAD API requiring only the corresponding docstring. On Figure 5 (left) and Figure 7 (supplementary) we showcase examples of our method utilizing the CAD commands *Revolution* and *Fillet* that are not included in existing datasets [61]. **Real-world use cases:** Tool augmentation allows interaction with multimodal inputs such as sketches and 3D scans. Figure 5 (center) showcases CAD-Assistant’s ability to process 3D scans along with textual queries to extract cross-sections, parameterize features, and reverse engineer CAD models from scans. In Figure 5 (right), the VLLM planner determines to semantically interpret drawn operation

directly without utilizing additional CAD-specific tools for fulfilling user requests. Note that generated FreeCAD code is interpretable, editable and easily extendable.

5. Conclusion

In this work, we propose CAD-Assistant, a generic tool-augmented CAD agent using CAD-specific tools. Our framework responds to multimodal queries via generated actions that are executed in a python interpreter integrated with FreeCAD. We assess CAD-Assistant on diverse CAD benchmarks and demonstrate the potential of tool-augmented VLLMs in real-world CAD workflows.

6. Acknowledgements

The present work is supported by the National Research Fund (FNR), Luxembourg, under the BRIDGES2021/IS/16849599/FREE-3D project and by Artec3D.

References

- [1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *NeurIPS*, 2022. 2
- [2] Kamel Alrashedy, Pradyumna Tambwekar, Zulfiqar Haider Zaidi, Megan Langwasser, Wei Xu, and Matthew Gombolay. Generating cad code with vision-language models for 3d designs. In *The Thirteenth International Conference on Learning Representations*. 3
- [3] Anthropic. Introducing the next generation of claude. 2023. 2
- [4] Akshay Badagabettu, Sai Sravan Yarlagadda, and Amir Barati Farimani. Query2cad: Generating cad models using natural language queries. *ArXiv*, 2024. 3
- [5] Antoine Brière-Côté, Louis Rivest, and Roland Maranzana. Comparing 3d cad models: uses, methods, tools and perspectives. *Computer-Aided Design and Applications*, 2012. 1
- [6] Francesco Buonamici, Monica Carfagni, Rocco Furferi, Lapo Governi, Alessandro Lapini, and Yary Volpe. Reverse engineering modeling methods and tools: a survey. *Computer-Aided Design and Applications*, 2018. 1
- [7] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *ArXiv*, 2023. 3
- [8] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *ICCV*, 2021. 3
- [9] Kseniya Cherenkova, Elona Dupont, Anis Kacem, Ilya Arzhannikov, Gleb Gusev, and Djamila Aouada. Sepicnet: Sharp edges recovery by parametric inference of curves in 3d shapes. In *CVPRW*, 2023. 1
- [10] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *ArXiv*, 2021. 3
- [11] FreeCAD Community. Freecad, 2024. 1, 2, 3, 5
- [12] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Albert Li, Pascale Fung, and Steven C. H. Hoi. Instructblip: Towards general-purpose vision-language models with instruction-tuning. *ArXiv*, 2023. 2
- [13] Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. Transcad: A hierarchical transformer for cad sequence inference from point clouds. In *ECCV*, 2024. 1
- [14] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Sriniwasan H. Sengamedu, and Christos Faloutsos. Large language models (llms) on tabular data: Prediction, generation, and understanding - a survey. *ArXiv*, 2024. 5
- [15] FreeCAD Community. The oca file format. 5
- [16] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. *CVPR*, 2022. 3, 5
- [17] Eunji Hong, Minh Hieu Nguyen, Mikaela Angelina Uy, and Minhyuk Sung. Mv2cyl: Reconstructing 3d extrusion cylinders from multi-view images. *NeurIPS*, 2024. 1
- [18] Cheng-Yu Hsieh, Sibe Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander J. Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *ArXiv*, 2023. 4
- [19] Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke S. Zettlemoyer, Noah A. Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *ArXiv*, 2024. 2, 3, 5
- [20] Autodesk Inc. Dxf reference, 2012. 5
- [21] Ahmet Serdar Karadeniz, Dimitrios Mallis, Nesryne Mejri, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Davinci: A single-stage architecture for constrained cad sketch inference. In *BMVC*, 2024. 1, 5, 7, 2
- [22] Ahmet Serdar Karadeniz, Dimitrios Mallis, Nesryne Mejri, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Picasso: A feed-forward framework for parametric inference of cad sketches via rendering self-supervision. In *WACV*, 2025. 1, 5, 6, 2
- [23] Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *CVPR*, 2024. 1
- [24] Claudius Kienle, Benjamin Alt, Darko Katic, and Rainer Jäkel. Querycad: Grounded question answering for cad models. *ArXiv*, 2024. 3
- [25] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *ICCV*, 2023. 3
- [26] Milin Kodnongbua, Benjamin Jones, Maaz Bin Safeer Ahmad, Vladimir Kim, and Adriana Schulz. Reparamcad: Zero-shot cad re-parameterization for interactive manipulation. In *SIGGRAPH Asia*, 2023. 1, 3
- [27] Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. In *Annual Meeting of the Association for Computational Linguistics*, 2021. 3
- [28] Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, 2022. 2
- [29] Xueyang Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. CAD translator: An effective drive for text to 3d parametric computer-aided design generative modeling. In *ACM Multimedia 2024*, 2024. 2

- [30] Xingang Li, Yuewan Sun, and Zhenghui Sha. Llm4cad: Multi-modal large language models for 3d computer-aided design generation. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2024. 3
- [31] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023. 2
- [32] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *CVPR*, 2024. 2
- [33] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *NeurIPS*, 2022. 2, 3
- [34] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *NeurIPS*, 2023. 2, 3, 5
- [35] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and A. Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *ICLR*, 2023. 2, 3
- [36] Liane Makatura, Michael Foshey, Bohan Wang, Felix Hahnlein, Pingchuan Ma, Bolei Deng, Megan Tjandrasuwita, Andrew Everett Spielberg, Crystal Elaine Owens, Peter 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? *ArXiv*, 2023. 2
- [37] Dimitrios Mallis, Ali Sk Aziz, Elona Dupont, Kseniya Cherenkova, Ahmet Serdar Karadeniz, Mohammad Sadil Khan, Anis Kacem, Gleb Gusev, and Djamila Aouada. Sharp challenge 2023: Solving cad history and parameters recovery from point clouds and 3d scans. overview, datasets, metrics, and baselines. In *CVPRW*, 2023. 1, 6, 2
- [38] Meta. The llama 3 herd of models. *ArXiv*, 2024. 2
- [39] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Ouyang Long, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback. *ArXiv*, 2021. 3
- [40] OpenAI. Gpt-4 technical report. 2023. 2, 3
- [41] Jeffrey M. Otey, Manuel Contero, and Jorge D. Camba. Revisiting the design intent concept in the context of mechanical cad education. *Computer-aided Design and Applications*, 2018. 6
- [42] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *ArXiv*, 2022. 3
- [43] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *ArXiv*, 2023. 3
- [44] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, Yi Ren Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shi Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bo Li, Ziwei Tang, Jing Yi, Yu Zhu, Zhenning Dai, Lan Yan, Xin Cong, Ya-Ting Lu, Weilin Zhao, Yuxiang Huang, Jun-Han Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. *ArXiv*, 2023. 3
- [45] Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Z. Xiao, Katherine M. Collins, Joshua B. Tenenbaum, Adrian Weller, Michael J. Black, and Bernhard Schölkopf. Can large language models understand symbolic graphics programs? *ArXiv*, 2024. 2, 3, 5, 6, 1
- [46] Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 2019. 3
- [47] Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-recode: Reverse engineering cad code from point clouds. *ArXiv*, 2024. 1
- [48] Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. In *ICMLW*, 2020. 2, 3, 5, 6, 7
- [49] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric cad sketches. In *ICLR*, 2022. 1, 2, 6, 7
- [50] Pratyusha Sharma, Tamar Rott Shaham, Manel Baradad, Stephanie Fu, Adrian Rodriguez-Munoz, Shivam Duggal, Phillip Isola, and Antonio Torralba. A vision check-up for language models. *CVPR*, 2024. 2, 3, 5
- [51] Yongliang Shen, Kaitao Song, Xu Tan, Dong Sheng Li, Weiming Lu, and Yue Ting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *ArXiv*, 2023. 2, 3
- [52] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In *Conference on Empirical Methods in Natural Language Processing*, 2021. 3
- [53] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. *International Conference on Web Search and Data Mining*, 2023. 6
- [54] D'idaç Sur'is, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *ICCV*, 2023. 2, 3, 5
- [55] Mikaela Angelina Uy, Yen-Yu Chang, Minhyuk Sung, Purvi Goel, Joseph G Lambourne, Tolga Birdal, and Leonidas J Guibas. Point2cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. In *CVPR*, 2022. 1
- [56] Chenfei Wu, Sheng-Kai Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *ArXiv*, 2023. 2, 3
- [57] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun

- Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *ArXiv*, abs/2308.08155, 2023. 1
- [58] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *CVPR*, 2021. 1, 2
- [59] Sifan Wu, Amir Hosein Khasahmadi, Mor Katz, Pradeep Kumar Jayaraman, Yewen Pu, Karl D.D. Willis, and Bang Liu. Cadvlm: Bridging language and vision in the generation of parametric cad sketches. *ECCV*, 2024. 3
- [60] Sifan Wu, Amir Hosein Khasahmadi, Mor Katz, Pradeep Kumar Jayaraman, Yewen Pu, Karl D.D. Willis, and Bang Liu. Cad-llm: Large language model for cad generation. 2024. 3
- [61] Peng Xu, Timothy M Hospedales, Qiyue Yin, Yi-Zhe Song, Tao Xiang, and Liang Wang. Deep learning for free-hand sketch: A survey. *IEEE TPAMI*, 2022. 1, 7, 8, 2
- [62] Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. In *ICML*, pages 24698–24724. PMLR, 2022. 2
- [63] Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable cad model generation. *ICML*, 2023. 2
- [64] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. In *arXiv*, 2023. 1
- [65] Yuezhi Yang and Hao Pan. Discovering design concepts for cad sketches. *arXiv*, 2022. 6
- [66] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *ArXiv*, 2023. 3
- [67] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *NIPS*, 2018. 3
- [68] Yang You, Mikaela Angelina Uy, Jiaqi Han, Rahul Thomas, Haotong Zhang, Suyu You, and Leonidas Guibas. Img2cad: Reverse engineering 3d cad models from images through vlm-assisted conditional factorization. *ArXiv*, 2024. 1, 3
- [69] Haocheng Yuan, Jing Xu, Hao Pan, Adrien Bousseau, Niloy J. Mitra, and Changjian Li. Cadtalk: An algorithm and benchmark for semantic commenting of cad programs. *CVPR*, 2024. 3
- [70] Andy Zeng, Adrian S. Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aavek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Peter R. Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *ArXiv*, 2022. 3
- [71] Yingzhong Zhang and Xiaofang Luo. Design intent information exchange of feature-based cad models. *2009 WRI World Congress on Computer Science and Information Engineering*, 2009. 6
- [72] Xiangyu Zhu, Dong Du, Weikai Chen, Zhiyou Zhao, Yinyu Nie, and Xiaoguang Han. Nerve: Neural volumetric edges for parametric curve extraction from point cloud. In *CVPR*, 2023. 1