CAD-Llama: Leveraging Large Language Models for Computer-Aided Design Parametric 3D Model Generation

Supplementary Material

1. Overview

In the supplementary material, we put forward some details about the data selection and method design. Cost analysis as well as extra experiment results are also put forward. The remaining parts are organized as follows.

- First, we provide a cost analysis on both GPU resource and GPT-4o tokens.
- Then we illustrate the format of our CAD code used throughout the pretraining and instruction tuning stage.
- After that we introduce the hierarchical annotation pipeline in detail with respect to CAD components, image extractor and two-stage prompting strategy.
- Finally we provide extra experiment results both quantitatively and qualitatively.

2. Training Cost and GPT-4o Token Cost

Both SPCC-adaptive pretraining and instruction tuning stages are conducted on 4 A100 GPUs. Table 1 summarizes the computational costs and token consumption for these stages. For generating finetuning data, during the SPCC-adaptive pretraining stage, altogether 70 million tokens are required to comprehend the image and generate prompts hierarchically. During the instruction tuning stage, 6 million tokens are used to generate instruction data. For the consumption of GPUs, SPCC-adaptive pretraining requires 38 A100-GPU hours and processes GPT-4o 70M tokens, while instruction tuning requires 12 A100-GPU hours and processes GPT-4o 6M tokens. This demonstrates that our model achieves efficient training with limited computational resources. Notably, during the Instruction Tuning phase, the model adapts effectively to various downstream tasks using only a small amount of data and training time.

Stage	A100-GPU Hours	Tokens (GPT-4o)
SPCC-Adaptive Pretraining	38	70M
Instruction Tuning	12	6M

Table 1. Training costs and token consumption during the two training stages. Tokens are used for prompt generation in each stage.

3. Details of CAD Code Formatting

We follow the annotations of DeepCAD [5] dataset and denote the components of the CAD command sequence. The complete set of command parameters is defined as $p_i = [x, y, \alpha, f, r, \theta, \phi, \gamma, p_x, p_y, p_z, s, e_1, e_2, b, u]$. We normalize and quantize these parameters as follows: (1) For discrete coordinate parameters, including the sketch plane origin (p_x, p_y, p_z), extrusion distances (e_1, e_2), curve endpoint coordinates (x, y), and the circle radius r, we quantize all continuous parameters into 256 levels, represent them with 8-bit integers, and recenter the origin from $(128, 128)$ to $(0, 0)$ for a more intuitive representation of scale. (2) For angular parameters, including the sketch orientation angles (θ, ϕ, γ) within the range $[-\pi, \pi]$ and the arc's sweep angle α within $[0, 2\pi]$, we use discrete values within the ranges $[-180, 180]$ and $[0, 360]$ degrees, respectively. (3) The sketch profile scale s is constrained within the range $[0, 2]$, while the boolean operation type b can take one of the following values: *new body*, *join*, *cut*, or *intersect*. The extrusion type u denotes one of three configurations: *one-sided*, *symmetric*, or *two-sided*. These parameters are utilized in their original forms. (4) The arc's counterclockwise flag f is a binary indicator, which we represent as either True or False.

For converting the annotation of CAD construction sequence into a LLM-friendly format, we further extract the hierarchy of CAD construction sequences and organize them into python-like pseudocode. In particular, the SOL and EOS commands are abstracted as an object `Loop()` and an ending comment `# End of code`, respectively. Other commands, such as *Arc*, *Line*, *Circle*, and *Extrude*, are represented as function calls with corresponding parameters as inputs of the function. Detailed examples are illustrated in Figure 2 and 3.

4. Details of Hierarchical Annotation Pipeline

4.1. Definition of CAD Component

In our definition, a CAD model consists of one or more components. Typically, a single sketch-extrude pair is treated as an individual component. However, when multiple identical sketch-extrude pairs occur consecutively in a CAD sequence, such as 10 cylinders uniformly distributed in a circular arrangement, describing each pair individually leads to redundancy and poses challenges for vision-language models (VLMs) in accurately capturing such repetitive structures.

To address this, when identical sketch-extrude pairs occur consecutively and their count exceeds a specified threshold (set to 3 in our experiments), we collectively define them as a single component. Otherwise, each sketch-

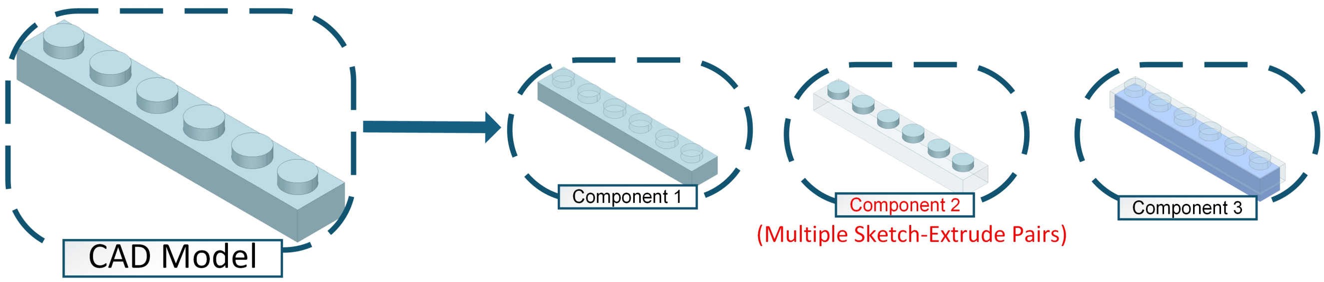


Figure 1. Illustration of defining a single component from consecutive equivalent sketch-extrude pairs based on specified criteria. Note that a large proportion of bottom cube in the final CAD model has been cut out, as is shown in component 3.

extrude pair is treated as an individual component. The equivalence of two sketch-extrude pairs is determined based on the following criteria: all commands and parameters must match, except for the sketch plane origin parameters (p_x, p_y, p_z). As illustrated in Figure 1, the second component comprises multiple sketch-extrude pairs.

4.2. Annotation Image Generation Pipeline

The hierarchical annotation pipeline contains two stages. Different images are fed into the VLM in different stages. We propose two kinds of image extractors which extracts different features of the CAD model, namely **Components Images Extractor** and **Outlines Images Extractor**, as shown in Figure 3 in the main paper. Both of them are python scripts rendering CAD construction sequences using PythonOCC [1] (Python version of OCCT) while focusing on different aspects of a single model. Taking the *i*-th component of the *j*-th CAD model $\mathcal{D}_j$ as an example, in the first stage, we use the **Components Images Extractor** and obtain the component image I_j^i and its corresponding 2D sketch image $\hat{I}_j^i$. Specifically, I_j^i is rendered from the default viewpoint by extracting the component's CAD command, while $\hat{I}_j^i$ is obtained by rendering the corresponding 2D sketch commands. In the second stage, we use the **Outlines Images Extractor** and obtain the outline image $\dot{I}_j^i$, achieved by increasing the transparency of other components (set to 0.85 in our experiments) while keeping the target component's transparency unchanged during rendering; if the target component is used for *cutting*, it is rendered in blue, as illustrated by component 3 in Figure 1.

4.3. Two Stage Prompting Methods.

In this part we provide the detailed prompt used in Section 4.2. In particular, two prompts are adopted where **prompt1** is for obtaining descriptions of individual components and **prompt2** is for acquiring both overall descriptions and component names. In **prompt1**, to enable GPT-4o to generate more detailed descriptions, we provide additional information that includes extrusion direction, extrusion length, and quantity information. The extrusion direction is included only when the CAD model is extruded in a specific direction, such as up, down, left, right, front, or back. We observed that over 95% of the extrusion directions in the DeepCAD [5] dataset fall within these categories. Quantity information is added only when a component contains multiple sketch-extrude pairs (see Section 4.1), which helps mitigate the hallucination phenomenon in VLM. The specific content of the two prompts is as follows:

Prompt1: *Background: The user now has a CAD model, which is formed by extruding a sketch. User input: The user will input two pictures, the first is the sketch, and the second is the CAD model after the sketch is extruded. Task: Describe the CAD model. Please describe the sketch in detail first, include the additional information in the description and output the final description result as a single line. Additional information: {Extrusion direction and length information, Number information} Examples: {Two Description Example}*

Prompt2: *A CAD model may consist of multiple modules. Each module constitutes a part of the model, which can be a solid or a feature used for cutting, such as creating a hole. The user has a CAD consisting of {num_parts} modules. The user will input {num_parts+1} pictures, the first image is the original CAD model, followed by {num_parts} images where each module is rendered with enhanced highlighting. These modules collectively form the original CAD model. Modules used for cutting are highlighted in blue. The subsequent description explains each of the four modules individually, following the order presented in the module images: {Component Descriptions} Task: You need to output three lines, Line 1: A concise description of the overall macro of CAD based on first image. Line 2: A detailed description that includes the specific characteristics of each of the {num_parts} modules mentioned above, as well as the process by which they are assembled based on all provided images and component descriptions. Line 3: Short names for {num_parts} modules. Example: {Two Description Example}*

5. Details of Experiments

5.1. Prompts Used in Baseline Methods

We provide the detailed prompt used in the baseline methods (GPT-4, GPT-3.5, LLaMA3, and Mistral) in Figure 5, where {task_definition} specifies the task instructions.

5.2. Ablation Study Details in Main Experiment

In the ablation study of the main experiment, we explored the impact of using different CAD representations for pretraining on the Text-to-CAD task. The evaluation methods are categorized based on whether the CAD data is in code format or raw sequences, and whether hierarchical or single descriptions are used. The single description $\mathcal{SD}$ of the CAD model $\mathcal{D}_j$ is defined as:

$$\mathcal{SD} = \text{concat}\{\mathcal{A}_j, \mathcal{T}_j, \text{“Parts description:”}, D_j\}$$

where $D_j = \text{concat}\{\{\mathcal{S}_j^1, \mathcal{I}_j^1\}, \{\mathcal{S}_j^2, \mathcal{I}_j^2\}, \ldots, \{\mathcal{S}_j^k, \mathcal{I}_j^k\}\}$ represent the full description of all k components of $\mathcal{D}_j$.

Tasks	Metric	*w/o* ICP	*with* ICP
Text-to-CAD	ACC_{cmd}	79.89	**80.41**
	ACC_{param}	59.04	**59.09**
Add	ACC_{cmd}	77.73	**79.41**
	ACC_{param}	62.16	**63.09**
Delete	EM	80.91	**81.93**

Table 2. Performance comparison of CAD-related tasks with and without In-Context Pretraining (ICP). The results show that ICP improves performance across all tasks

5.3. Ablation Studies on Pretraining Method

This section presents a simple ablation study on several CAD-related tasks to validate the effectiveness of In-Context Pretraining (ICP) [3] in enhancing CAD-Llama-INS performance on downstream tasks. ICP is a method that groups related documents within the same input context, encouraging LLMs to read and reason across document boundaries. Similar to [3], we used a pretrained CLIP [2] model to encode CAD images and group similar CADs for pretraining based on their cosine similarity. As shown in Table 2, ICP enhances the performance of downstream editing tasks, such as add and delete, by enabling LLMs to better capture the distinctions between different CAD structures during pretraining. This improved understanding allows the model to more effectively handle precise modifications required in these tasks. Additionally, ICL contributes to a marginal improvement in the Text-to-CAD task.

Dataset	$ACC_T \uparrow$	MCD$\downarrow$	MMD$\downarrow$	JSD$\downarrow$
DeepCAD	84.72	10.53	1.54	3.59
Fusion360	78.35	23.06	1.98	6.01

Table 3. Our model, CAD-Llama-INS, trained exclusively on the DeepCAD dataset, demonstrates strong generalization capabilities on the Fusion360 dataset in the Text-to-CAD task.

5.4. Cross Dataset Generalization

To further evaluate the generalization ability of CAD-Llama-INS, we conducted experiments on the test set of the Fusion 360 [4] dataset for the Text-to-CAD task. Similar to DeepCAD [5], the Fusion 360 dataset also contains CAD construction sequences. We employed the hierarchical annotation pipeline to generate descriptions for the Fusion 360 dataset. These descriptions are used to prompt CAD-Llama-INS, which was pre-trained and fine-tuned exclusively on the DeepCAD dataset, to produce corresponding CAD models. The experimental results, as shown in Table 3, demonstrate that CAD-Llama-INS achieves strong generalization performance, achieving comparable or superior results on the Fusion 360 dataset despite being trained solely on DeepCAD. This highlights the effectiveness of our approach in adapting to new datasets. A qualitative analysis is also conducted, as illustrated in Figure 6. Based on textual prompts, CAD-Llama-INS demonstrates the capability to generate CAD models that closely align with the ground truth.

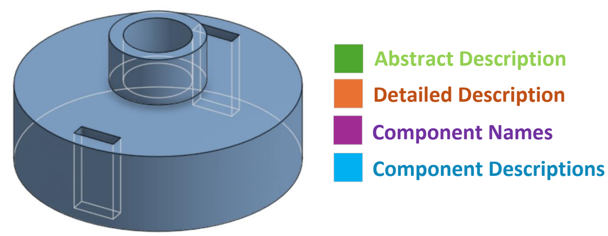
5.5. Qualitative results

To comprehensively evaluate the performance of our approach, we provide qualitative results across multiple tasks. Specifically, qualitative results for text-to-CAD generation are illustrated in Figure 7 to Figure 17. Results for captioning tasks are presented in Figure 4, while results for unconditional generation are shown in Figure 18. Additionally, results for multi-task evaluation, encompassing the process from initial construction to iterative refinement, are shown in Figures 13 to 17.

5.6. Examples of Failure Cases

Our experimental results also show some limitations of our method, in some cases there are parameter generation errors and mismatching between the input text instruction and the generated CAD command sequences, Figure 19 illustrates some failure cases of Text-to-CAD generation.

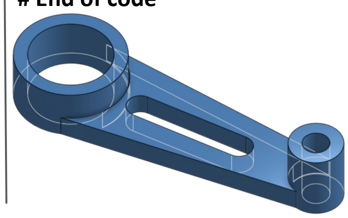
```
# Description of the model: A cylindrical disc with two rectangular cutouts on its top surface is combined with a thick-walled cylindrical ring with a hollow center. The thick-walled cylindrical ring is placed on top of the cylindrical disc, aligning the hollow center of the ring with the center of the disc, with two rectangular cutouts on the top surface of the disc.
# Cylinder base: A cylindrical base is extruded upwards to a length of 59 units.
sketch0 = []
loop0_0 = Loop()
loop0_0.Circle(center=(48, 0),radius=48)
sketch0.add(loop0_0)
Extrude(sketch=sketch0, orientation_degrees=(0, 0, 0), origin=(-96, 0, 0), scale=2.0, extrude_distance=(59, 0), boolean_type='New', extent_type='One-sided')
# Cylindrical extension with hole: A cylindrical shape with a central circular hole is extruded upwards to a length of 35 units.
sketch1 = []
loop1_0 = Loop()
loop1_0 .Circle(center=(48, 0),radius=48)
sketch1.add(loop1_0)
loop1_1 = Loop()
loop1_1.Circle(center=(48, 0),radius=34)
sketch1.add(loop1_1)
Extrude(sketch=sketch1, orientation_degrees=(0, 0, 0), origin=(-32, 0, 61), scale=0.68, extrude_distance=(35, 0), boolean_type='Joining', extent_type='One-sided')
# Rectangular cutouts: Two rectangular cutouts, positioned symmetrically on the surface of the cylindrical base and extending partially through its thickness, are extruded upwards to a length of 60 units.
sketch2 = []
loop2_0= Loop()
loop2_0.Line(endpoint=(55, -42))
loop2_0.Line(endpoint=(55, -37))
loop2_0.Line(endpoint=(40, -37))
loop2_0.Line(endpoint=(40, -42))
sketch2.add(loop2_0)
loop2_1 = Loop()
loop2_1.Line(endpoint=(55, 37))
loop2_1.Line(endpoint=(55, 42))
loop2_1.Line(endpoint=(40, 42))
loop2_1.Line(endpoint=(40, 37))
sketch2.add(loop2_1)
Extrude(sketch=sketch2, orientation_degrees=(0, 0, 0), origin=(-96, 0, 0), scale=2.0, extrude_distance=(60, 0), boolean_type='Cutting', extent_type='One-sided')
# End of code
```




```
# Description of the model: A hexagonal prism with a central cylindrical hole running through its length, combined with two concentric cylindrical tubes arranged horizontally. The hexagonal prism with a central cylindrical hole is extruded backward longitudinally. Then, two concentric cylindrical tubes with central circular holes are extruded backward longitudinally and arranged horizontally to fit into the hexagonal prism.
# Hexagonal prism with cylindrical hole: A hexagonal prism with a central cylindrical hole running through its length is extruded backward to a length of 96 units.
sketch0 = []
loop0_0 = Loop()
loop0_0.Line(endpoint=(18, -45))
loop0_0.Line(endpoint=(66, -50))
loop0_0.Line(endpoint=(95, -14))
loop0_0.Line(endpoint=(77, 31))
loop0_0.Line(endpoint=(29, 36))
loop0_0.Line(endpoint=(0, 0))
sketch0.add(loop0_0)
loop0_1 = Loop()
loop0_1.Circle(center=(48, -7), radius=24)
sketch0.add(loop0_1)
Extrude(sketch=sketch0, orientation_degrees=(90, -90, 90), origin=(-51, 0, 8), scale=1.06, extrude_distance=(96, 0), boolean_type='New', extent_type='One-sided')
# Concentric cylindrical tubes: Two concentric cylindrical tubes with central circular holes are arranged horizontally and extruded backward to a length of 26 units.
sketch1 = []
loop1_0 = Loop()
loop1_0.Circle(center=(48, 0), radius=48)
sketch1.add(loop1_0)
loop1_1 = Loop()
loop1_1.Circle(center=(48, 0), radius=22)
sketch1.add(loop1_1)
Extrude(sketch=sketch1, orientation_degrees=(90, -90, 90), origin=(-62, 0, 0), scale=1.29, extrude_distance=(51, 0), boolean_type='Joining', extent_type='One-sided')
sketch2 = []
loop2_0 = Loop()
loop2_0.Circle(center=(48, 0), radius=48)
sketch2.add(loop2_0)
loop2_1 = Loop()
loop2_1.Circle(center=(48, 0), radius=30)
sketch2.add(loop2_1)
Extrude(sketch=sketch2, orientation_degrees=(90, -90, 90), origin=(-27, 0, 0), scale=0.57, extrude_distance=(26, 0), boolean_type='Joining', extent_type='One-sided')
# End of code
```

Figure 2. Examples of SPCC data representation, generated by our CAD-Llama-INS.

```
# Description of the model: A tapered rectangular prism featuring an elongated oval slot through the center, with semi-circular cutouts on opposite sides, which connected to two cylindrical shapes with central circular holes. The tapered rectangular plate is positioned horizontally, and the two cylindrical shapes of different sizes are placed vertically on the semicircular cutouts at each end of the plate, aligning their central circular holes with the cutouts.
# Rectangular plate with cutouts and slot: A rectangular plate featuring two large semicircular cutouts on opposite sides and a central elongated oval slot is extruded upwards to a length of 7 units.
sketch0 = []
loop = Loop()
loop.Line(endpoint=(95, 10))
loop.Arc(endpoint=(94, 29), degrees=180, counterclockwise=False)
loop.Line(endpoint=(0, 39))
loop.Arc(endpoint=(0, 0), degrees=168, counterclockwise=False)
sketch0.add(loop)
loop = Loop()
loop.Line(endpoint=(64, 14))
loop.Arc(endpoint=(64, 24), degrees=180, counterclockwise=True)
loop.Line(endpoint=(25, 24))
loop.Arc(endpoint=(25, 14), degrees=180, counterclockwise=True)
sketch0.add(loop)
Extrude(sketch=sketch0, orientation_degrees=(0, 0, 0), origin=(3, -17, 0), scale=0.89, extrude_distance=(7, 0), boolean_type='New', extent_type='One-sided')
# Large cylindrical Shape: A cylindrical shape with a central circular hole, resembling a thick washer, is extruded upwards to a length of 17 units.
sketch1 = []
loop = Loop()
loop.Circle(center=(48, 0),radius=48)
sketch1.add(loop)
loop = Loop()
loop.Circle(center=(48, 0),radius=36)
sketch1.add(loop)
Extrude(sketch=sketch1, orientation_degrees=(0, 0, 0), origin=(-17, 0, 0), scale=0.36, extrude_distance=(17, 0), boolean_type='Joining', extent_type='One-sided')
# Small Cylindrical Shape: A cylindrical shape, featuring a central circular hole on one end and resembling a thick washer, is extruded upwards to a length of 22 units.
sketch2 = []
loop = Loop()
loop.Circle(center=(48, 0),radius=48)
sketch2.add(loop)
loop = Loop()
loop.Circle(center=(48, 0),radius=24)
sketch2.add(loop)
Extrude(sketch=sketch2, orientation_degrees=(0, 0, 0), origin=(79, 0, 0), scale=0.18, extrude_distance=(22, 0), boolean_type='Joining', extent_type='One-sided')
# End of code
```



Abstract Description
Detailed Description
Component Names
Component Descriptions

```
# Description of the model: A central cylinder is surrounded by six evenly spaced cylindrical pillars arranged in a circular pattern. The central cylinder is extruded vertically, and the six evenly spaced cylindrical pillars are also extruded vertically and arranged in a circular pattern around the central cylinder.
# Central Cylinder: A cylindrical shape is extruded upwards to a length of 72 units.
sketch0 = []
loop0_0 = Loop()
loop0_0.Circle(center=(48, 0), radius=48)
sketch0.add(loop0_0)
Extrude(sketch=sketch0, orientation_degrees=(0, 0, 0), origin=(-60, 0, 0), scale=1.25, extrude_distance=(72, 0), boolean_type='New', extent_type='One-sided')
# Circular array of cylindrical pillars: Six evenly spaced cylindrical pillars, arranged in a circular pattern, extruded upwards to a length of 96 units.
sketch1 = []
loop1_0 = Loop()
loop1_0.Circle(center=(48, 0), radius=48)
sketch1.add(loop1_0)
Extrude(sketch=sketch1, orientation_degrees=(0, 0, 0), origin=(-10, 60, 0), scale=0.2, extrude_distance=(96, 0), boolean_type='New', extent_type='One-sided')
sketch2 = []
loop2_0 = Loop()
loop2_0.Circle(center=(48, 0), radius=47)
sketch2.add(loop2_0)
Extrude(sketch=sketch2, orientation_degrees=(0, 0, 0), origin=(-61, 30, 0), scale=0.2, extrude_distance=(96, 0), boolean_type='Joining', extent_type='One-sided')
sketch3 = []
loop3_0 = Loop()
loop3_0.Circle(center=(48, 0), radius=47)
sketch3.add(loop3_0)
Extrude(sketch=sketch3, orientation_degrees=(0, 0, 0), origin=(-61, -30, 0), scale=0.2, extrude_distance=(96, 0), boolean_type='Joining', extent_type='One-sided')
sketch4 = []
loop4_0 = Loop()
loop4_0.Circle(center=(48, 0), radius=48)
sketch4.add(loop4_0)
Extrude(sketch=sketch4, orientation_degrees=(0, 0, 0), origin=(-10, -60, 0), scale=0.2, extrude_distance=(96, 0), boolean_type='Joining', extent_type='One-sided')
sketch5 = []
loop5_0 = Loop()
loop5_0.Circle(center=(48, 0), radius=47)
sketch5.add(loop5_0)
Extrude(sketch=sketch5, orientation_degrees=(0, 0, 0), origin=(42, -30, 0), scale=0.2, extrude_distance=(96, 0), boolean_type='Joining', extent_type='One-sided')
sketch6 = []
loop6_0 = Loop()
loop6_0.Circle(center=(48, 0), radius=47)
sketch6.add(loop6_0)
Extrude(sketch=sketch6, orientation_degrees=(0, 0, 0), origin=(42, 30, 0), scale=0.2, extrude_distance=(96, 0), boolean_type='Joining', extent_type='One-sided')
# End of code
```

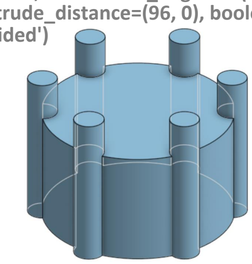


Figure 3. Examples of SPCC data representation, generated by our CAD-Llama-INS.

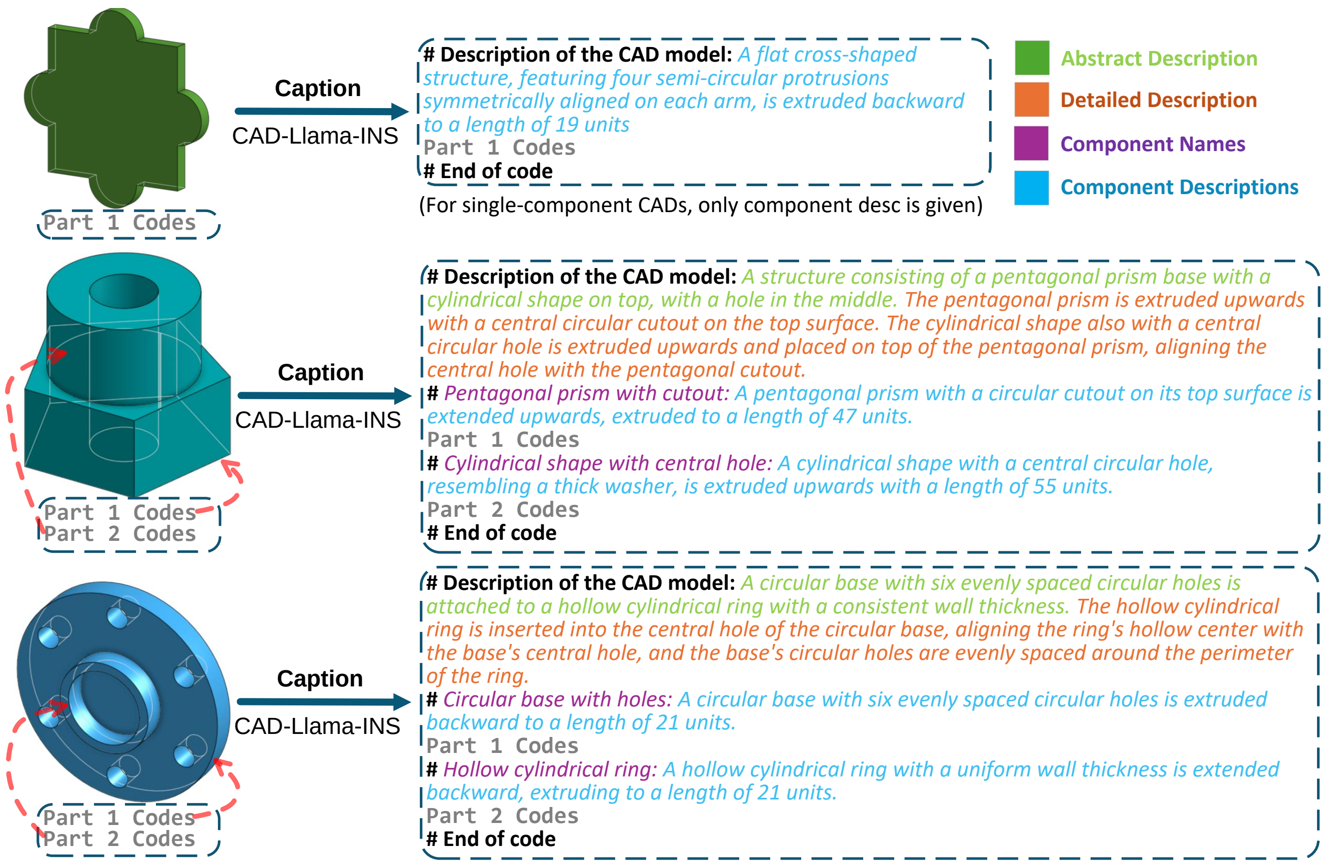


Figure 4. Examples of results from the Caption task, demonstrating the capabilities of CAD-Llama-INS in understanding the internal structure of raw CAD code and its geometric shapes.

Prompt for Baseline Methods (2-shot)

<background information>
Computer-Aided Design (CAD) is a technology used in engineering and manufacturing to create precise 2D and 3D models. In CAD modeling, common geometric operations include drawing lines (Line), arcs (Arc), and circles (Circle) to form sketches that define the shape of the object.
These sketches are usually composed of multiple loops, with each loop made up of various geometric entities like lines, arcs, and circles to form a closed surface. Once the sketch is created, an extrusion operation can be applied to extend the 2D profile into 3D space, thus forming a solid body. The extrusion process involves setting parameters such as the orientation of the sketch plane and the distance of extrusion.
In CAD systems, the creation of 3D objects usually begins with 2D sketches consisting of geometric entities like lines, arcs, and circles. Each of these entities is defined by specific parameters. The default starting point of each loop is (0,0).
`Line` command is determined by its `endpoint` coordinates.
`Arc` command requires its `endpoint`, a sweep angle (`degrees`), and a direction flag (`counterclockwise`) to indicate whether it is drawn clockwise or counterclockwise.
`Circle` command is defined by its `center` coordinates and `radius`. Once the sketch is complete.
`Extrude` command is used to transform the 2D sketch into a 3D object. The extrusion process involves parameters such as the orientation of the sketch plane, defined by three angles (`orientation_degrees`), and the sketch plane's position in 3D space (`origin`). The `scale` parameter controls the size of the sketch profile, while `extrude_distance` defines how far the sketch is extended in both directions. The `boolean_type` specifies how the new geometry interacts with the existing model, with options like 'New' (creating a new solid), 'Joining' (merging with the existing body), 'Cutting' (removing material), and 'Intersection' (keeping only the overlapping portion). The `extent_type` controls the extrusion direction, with options such as 'One-sided' (extending in one direction), 'Symmetric' (extending equally in both directions), or 'Two-sided' (with different distances on each side).
Note: All coordinate values are quantized to integers between 0 and 255, so your output coordinate value should be less than or equal to 255.
</background information>
<task>{task_definition}</task>
<example><input>{e1_input}</input><output>{e1_output}</output></example>
<example><input>{e2_input}</input><output>{e2_output}</output></example>

Figure 5. Detailed prompt used in the baseline methods (GPT-4, GPT-3.5, LLaMA and Mistrial).

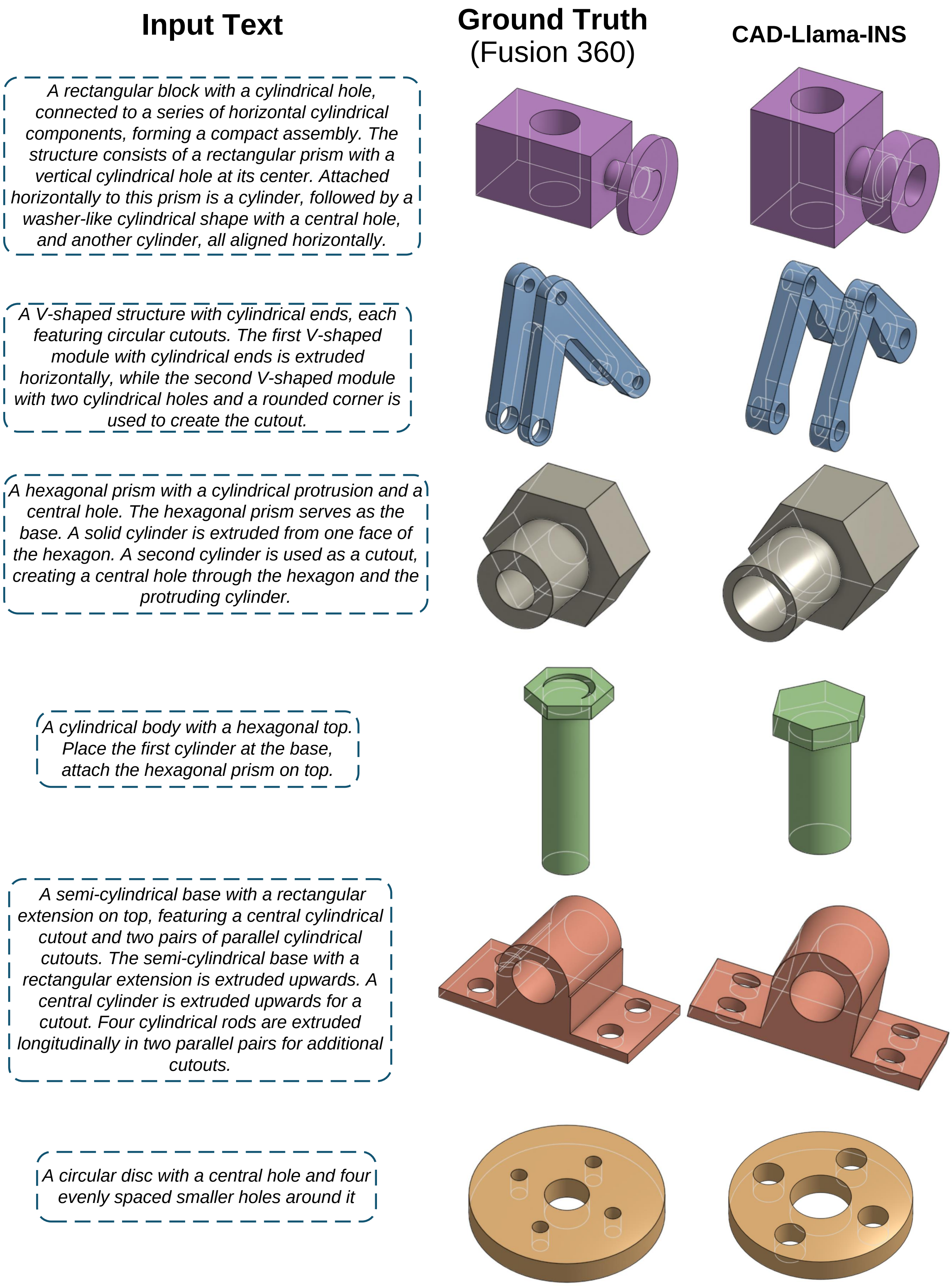
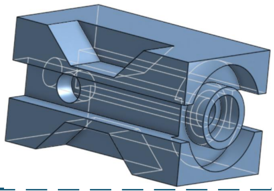
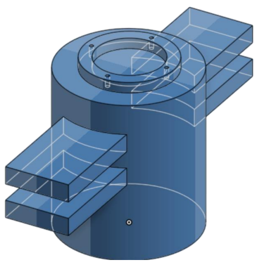


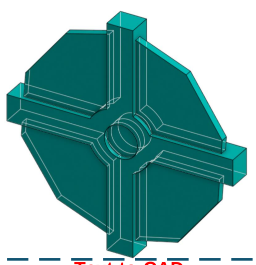
Figure 6. Comparison results of Text-to-CAD task on the Fusion 360 dataset.



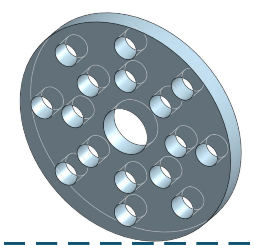
Text-to-CAD
A horizontally oriented rectangular block with symmetrical notches on opposite sides, forming angular cutouts. A central circular hole runs through the length of the block, with additional cylindrical cutouts intersecting it. On either end, there are concentric cylindrical features, including a hollow cylindrical tube with a slanted edge. These cylindrical elements are mirrored on both sides of the block. The angular cutouts are formed by triangular prisms intersecting the block at a 45-degree angle, creating a stepped appearance along its sides



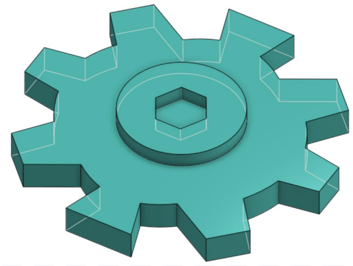
Text-to-CAD
The CAD creation integrates simplified complexity. A cylinder forms the core structure, providing foundational stability and rotational symmetry. Mounted laterally, a rectangular solid block with a horizontal groove runs along its length, aiding in the precise linear arrangement of associated parts. Finally, a circular ring with four smaller, equally spaced holes is securely affixed to the top of the cylinder, providing multiple anchoring points for securing or interfacing with other components.



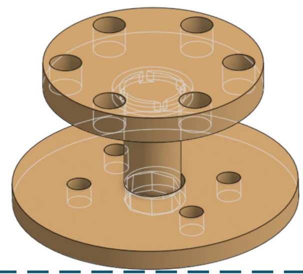
Text-to-CAD
A dynamic cross-sectional framework embedded with a dodecagonal pattern. The cross-shaped structure, combined with a dodecagonal prism, aligns its central circular hole with one of the prism's faces, effectively creating an intricate framework with elongated arms supporting the dodecagonal edges.



Text-to-CAD
Incorporating a circular plate with a large central hole and two sets of eight smaller circular holes arranged in a ring pattern, one set located closer to the center and the other set positioned near the outer edge, symmetrically distributed around the central hole.

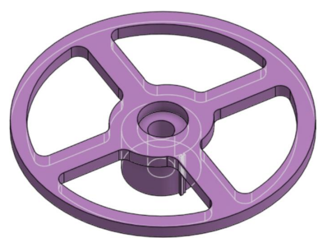


Text-to-CAD
The model features a circular base with eight evenly spaced trapezoidal teeth extending radially outward from its perimeter. At the center of the base is a circular disk, which contains a hexagonal hole passing through it.



Text-to-CAD
A circular disc with six holes is mounted on a cylindrical base, with a central octagonal support. The circular disc with six equally spaced holes is placed on top. Below it, a long main column serves as support, with a polygonal structure located beneath the column. A spherical ring with five holes is then placed around the base of the column.

Text-to-CAD
The base features a large rectangular hollow at its center on the bottom side. On top of the base, there are two symmetrical rectangular protrusions extending outward on either side. At the center of the base's top surface is a semi-circular concave feature that connects the structure above to the base. Rising from the base are two symmetrical sets of inclined planes, joined at the center by a smooth saddle-shaped curved surface. Four prismatic arms extend diagonally upward from the base, symmetrically positioned on either side.



Text-to-CAD
The foundation is a circular plate with a central solid hub and four equally spaced rectangular cutouts extending radially towards the outer edge, creating a cross-like pattern. Below this, a cylindrical tube formed by extruding a circular ring with a central hole is mounted around the hub.

Figure 7. Supplementary results of the Text-to-CAD task generated by CAD-Llama-INS based on text prompts.

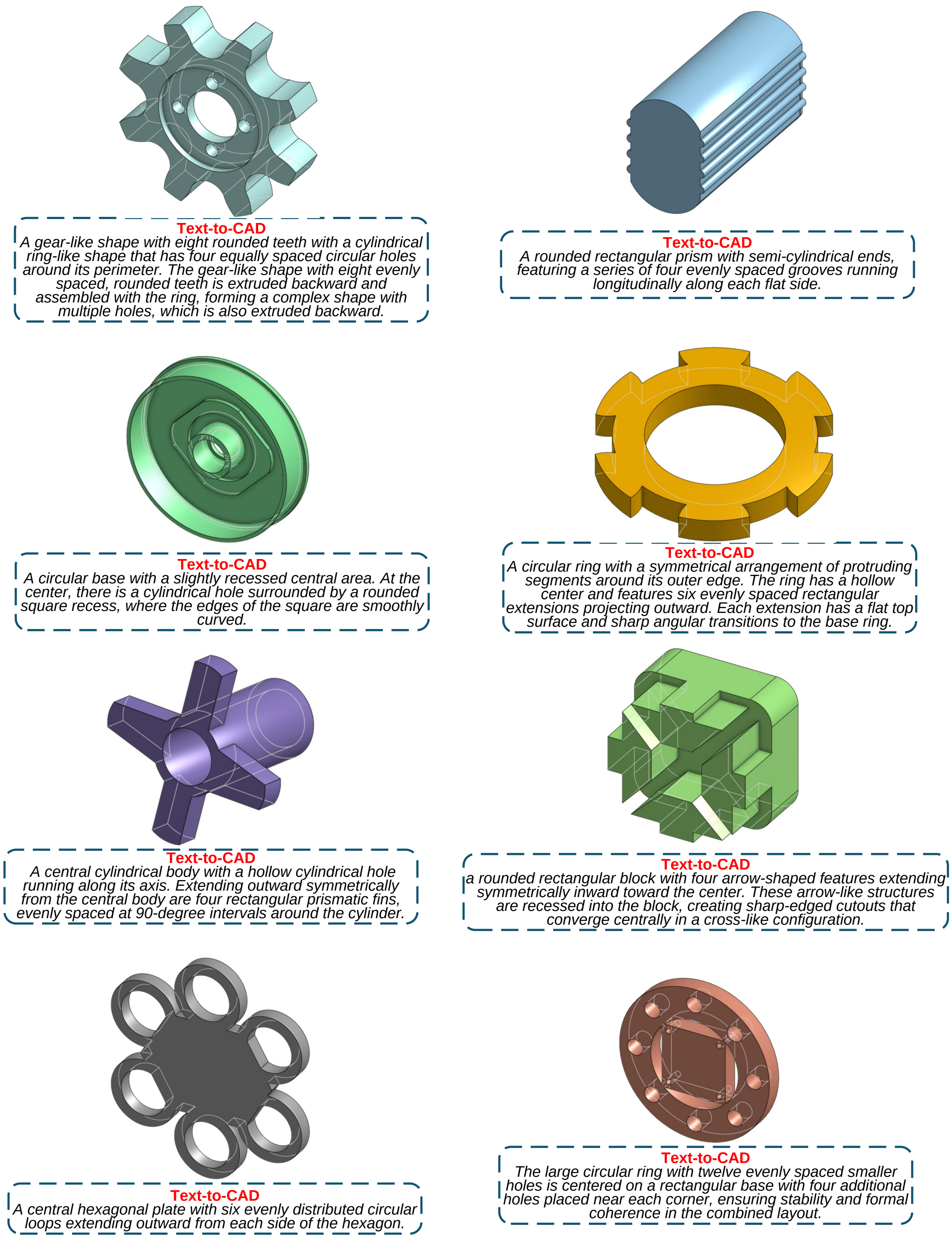


Figure 8. Supplementary results of the Text-to-CAD task generated by CAD-Llama-INS based on text prompts.

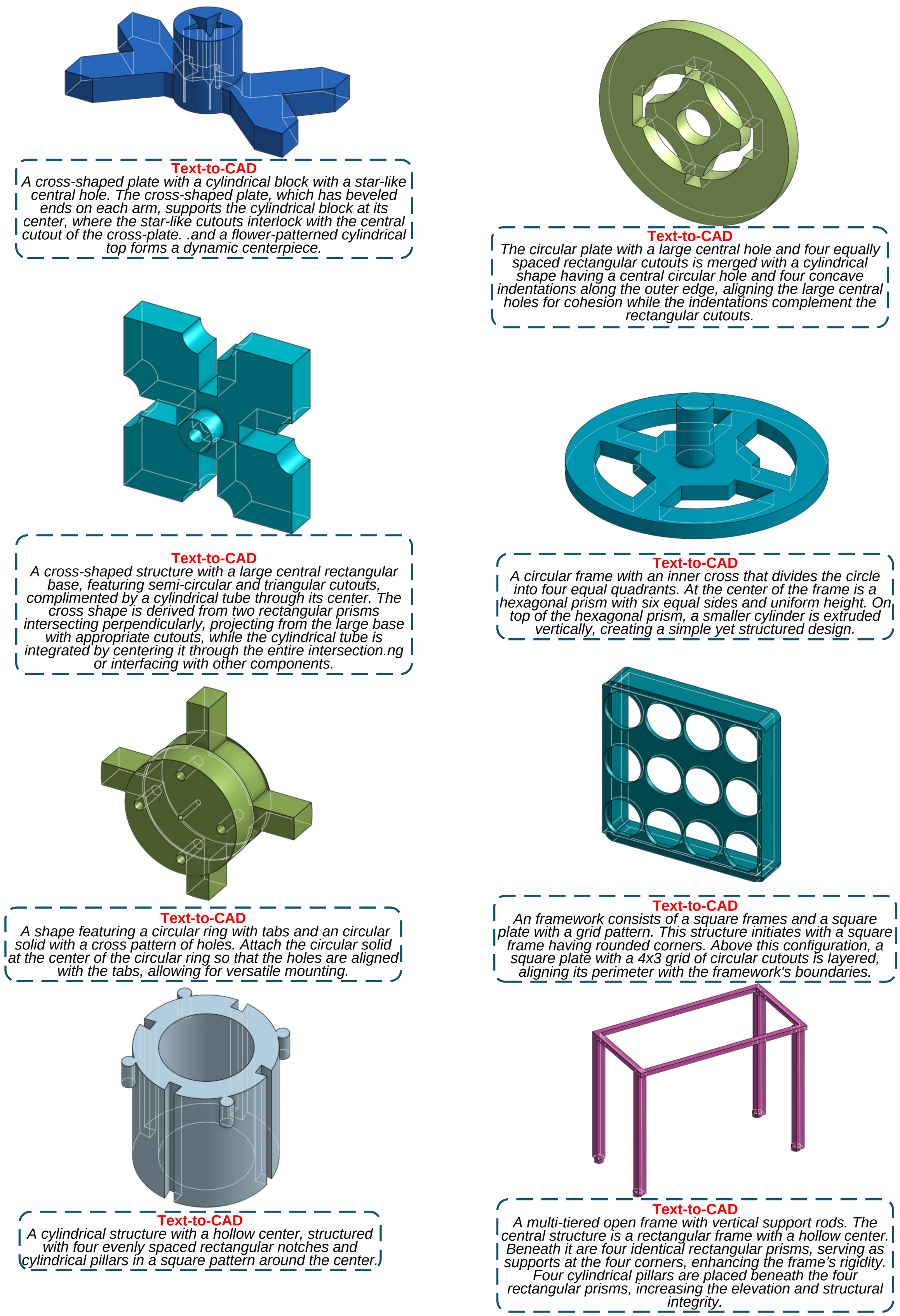
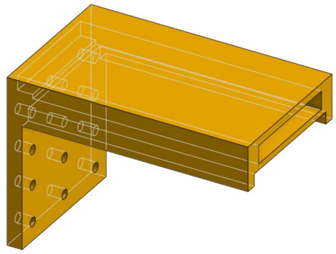
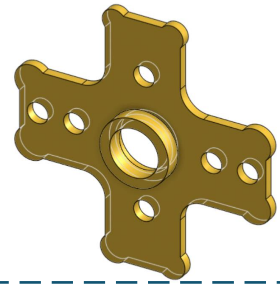


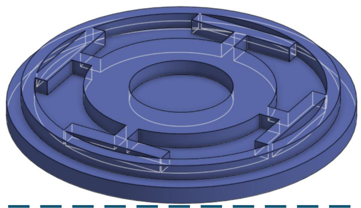
Figure 9. Supplementary results of the Text-to-CAD task generated by CAD-Llama-INS based on text prompts.



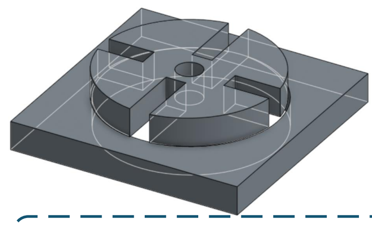
Text-to-CAD
A rectangular block with a large central rectangular cutout and two small rectangular notches at the bottom edge, enhanced by a square plate with twelve equally sized circular holes arranged in a circular pattern fixed onto the left side of the block.



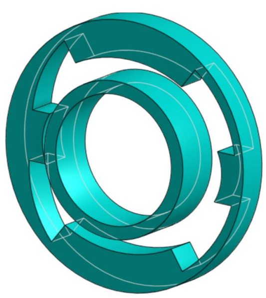
Text-to-CAD
An intricate structure combining a flat, symmetrical cross-shaped plate with rounded arms and a toroidal ring with a hollow circular center. The cross-shaped plate houses the toroidal ring in its center, with the circular holes near the ends of the arms providing attachment points.



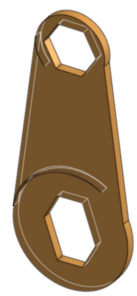
Text-to-CAD
A circular base with a central cylindrical extrusion, intersected by a circular ring and four perpendicular bars, each bar having rectangular blocks attached to their ends, placed on top of the cylindrical extrusion.



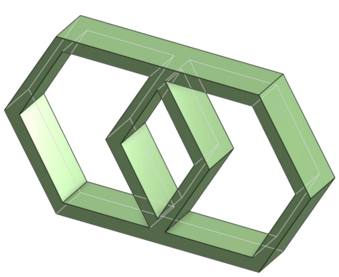
Text-to-CAD
A circular disc with a central hole and four evenly spaced rectangular cutouts near the perimeter on top of the cylindrical protrusion, aligning the central hole with the cylinder.



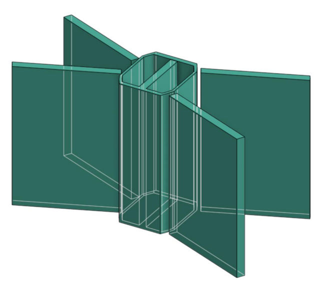
Text-to-CAD
Two concentric circular rings connected by evenly spaced rectangular supports. The outer ring is larger and serves as the primary boundary, while the inner ring is smaller and located centrally.



Text-to-CAD
Two cylindrical discs, each featuring a central hexagonal hole. The discs are connected by a tapered rectangular plate, forming an elongated shape. One disc is larger in diameter and positioned at one end of the plate, while the smaller disc is located at the opposite end.



Text-to-CAD
Two hollow hexagonal frames are horizontally staggered and stacked together. The frames are connected along a shared edge, creating an overlapping configuration. Each frame has a uniform rectangular cross-section, and their hollow interiors remain open.



Text-to-CAD
A central hollow rounded square prism with four flat faces. The hollow center is divided into two equal parts by a thin rectangular plate. Four rectangular plates extend outward from the faces of the prism in an X-like configuration

Figure 10. Supplementary results of the Text-to-CAD task generated by CAD-Llama-INS based on text prompts.

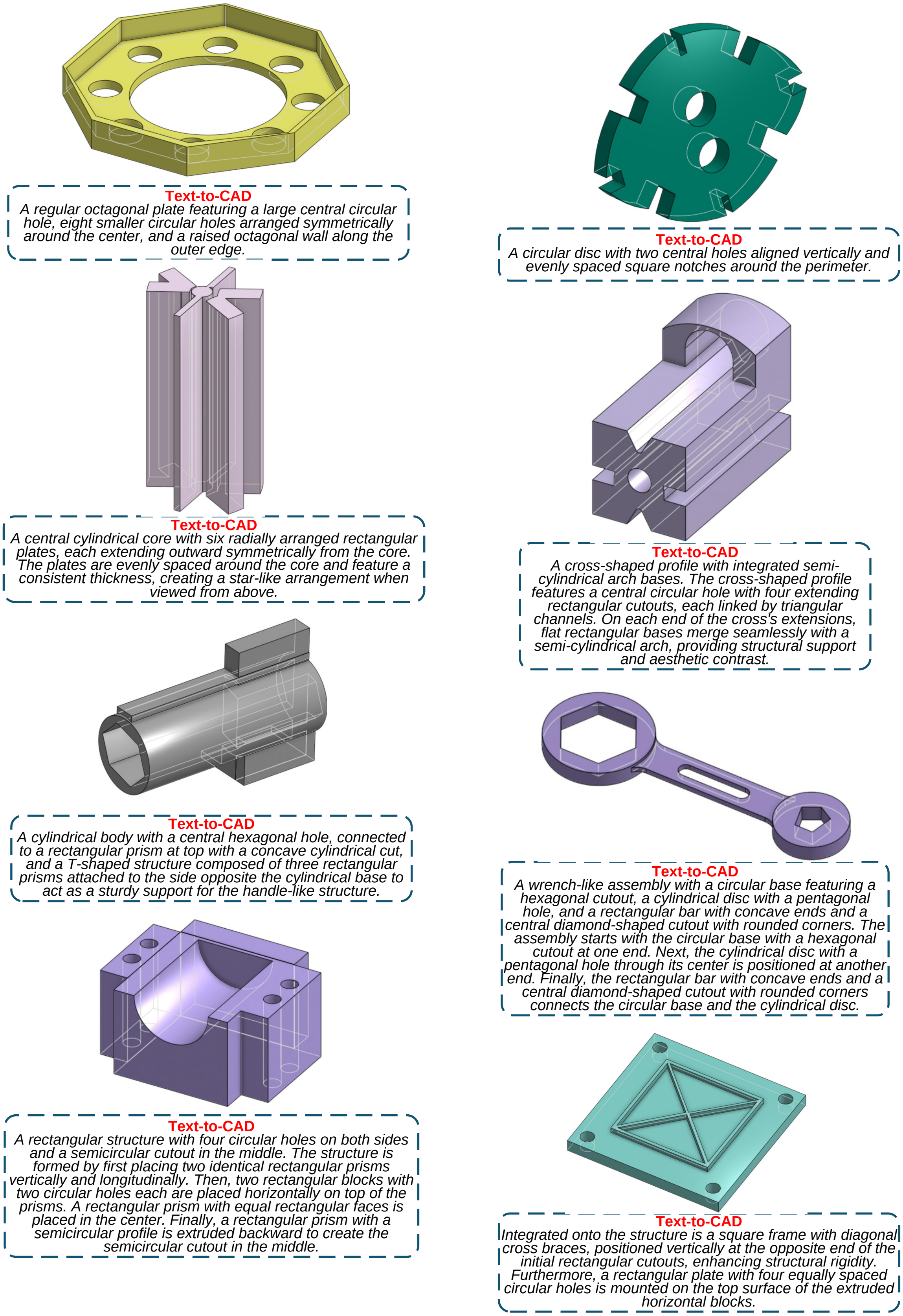


Figure 11. Supplementary results of the Text-to-CAD task generated by CAD-Llama-INS based on text prompts.

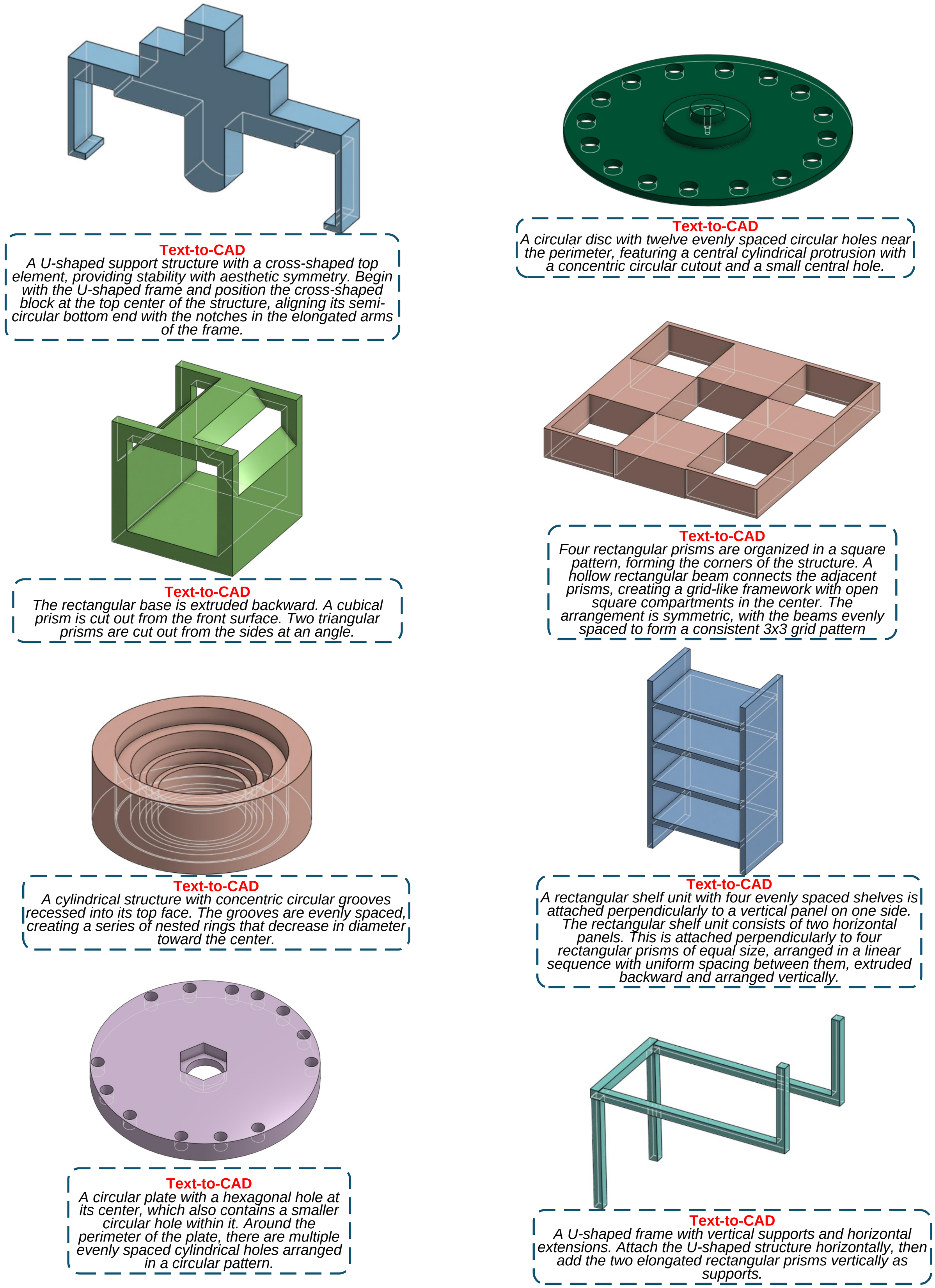


Figure 12. Supplementary results of the Text-to-CAD task generated by CAD-Llama-INS based on text prompts.

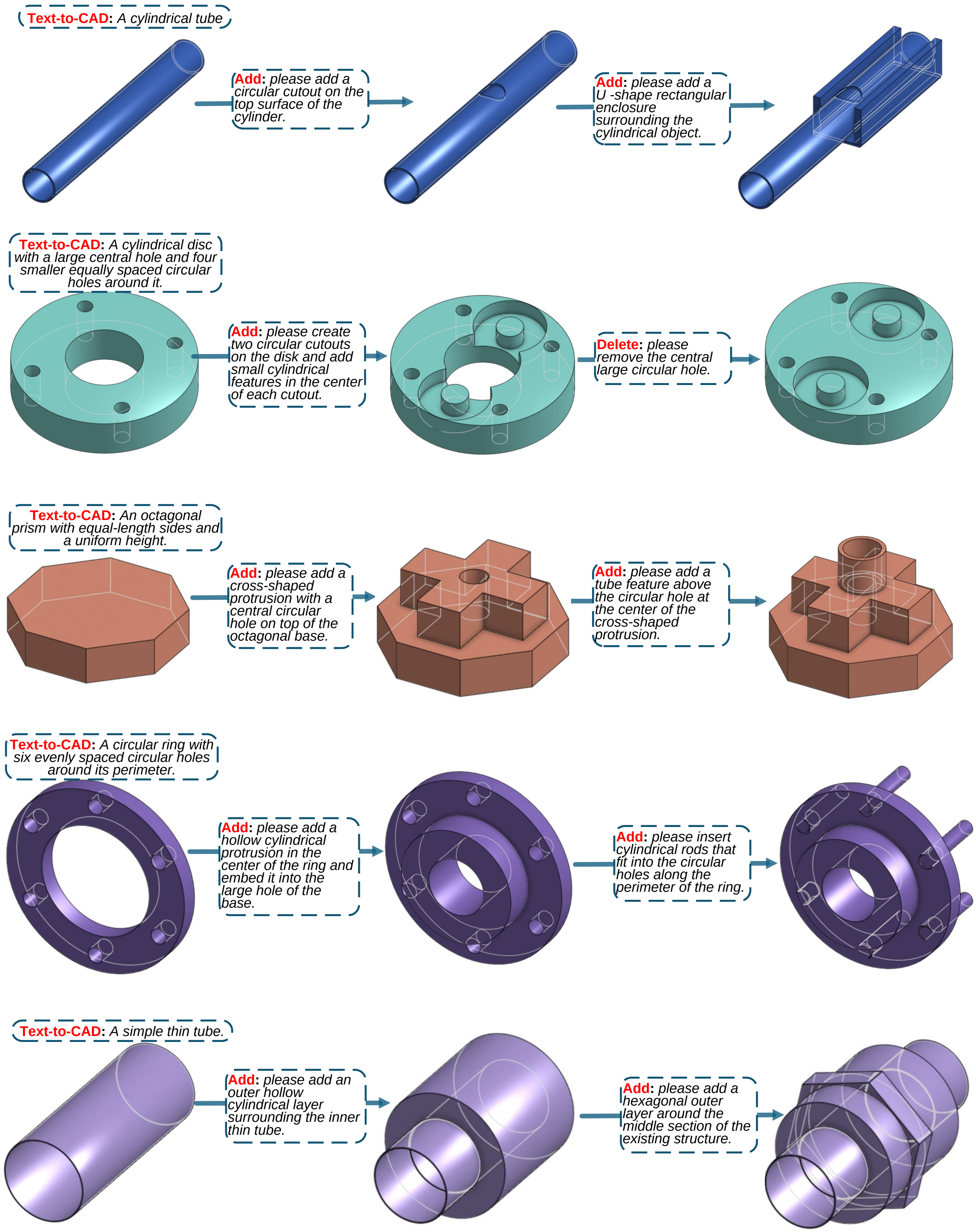


Figure 13. Supplementary working examples of Text-to-CAD, Delete, Add tasks using CAD-Llama-INS.

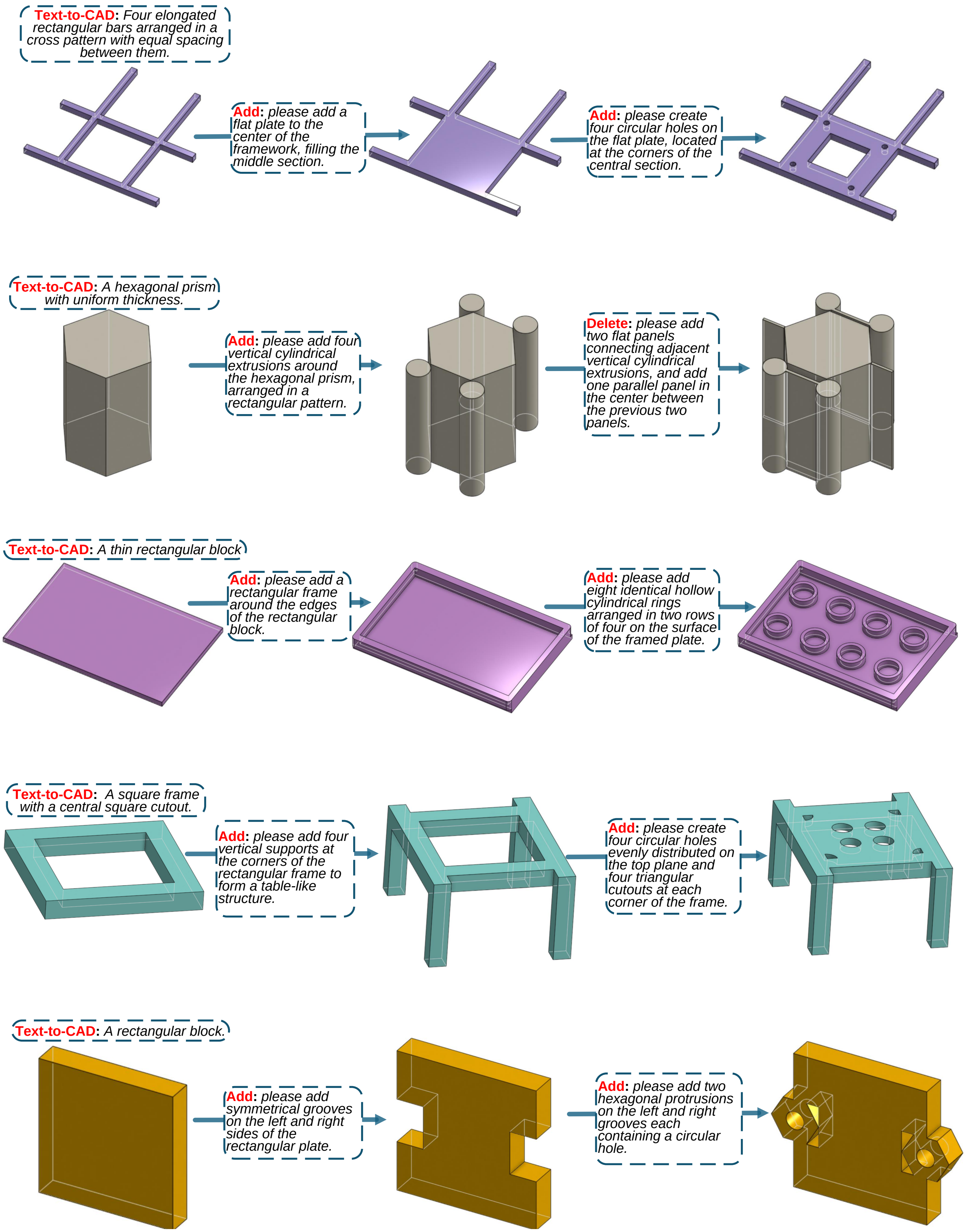


Figure 14. Supplementary working examples of Text-to-CAD, Delete, Add tasks using CAD-Llama-INS.

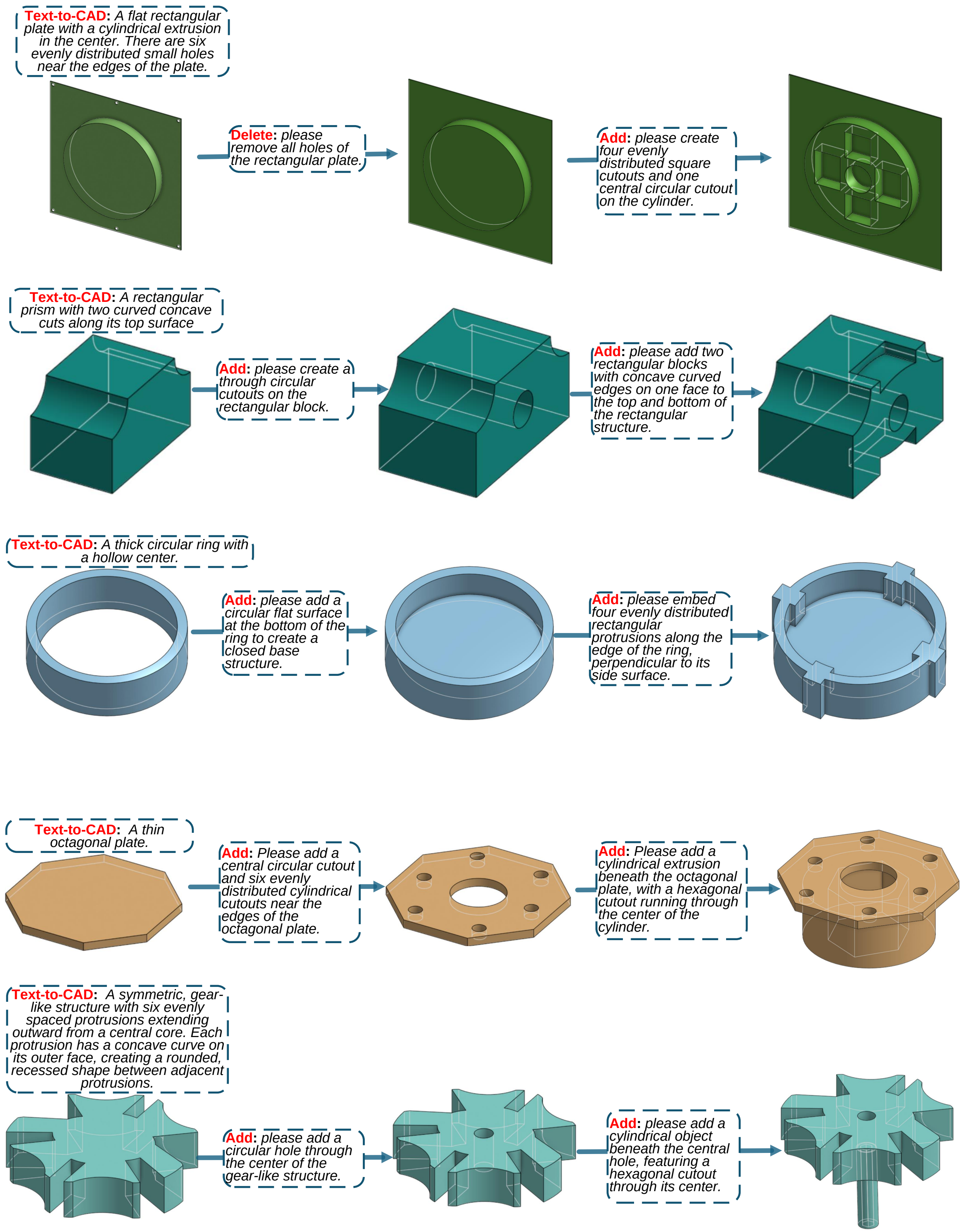


Figure 15. Supplementary working examples of Text-to-CAD, Delete, Add tasks using CAD-Llama-INS.

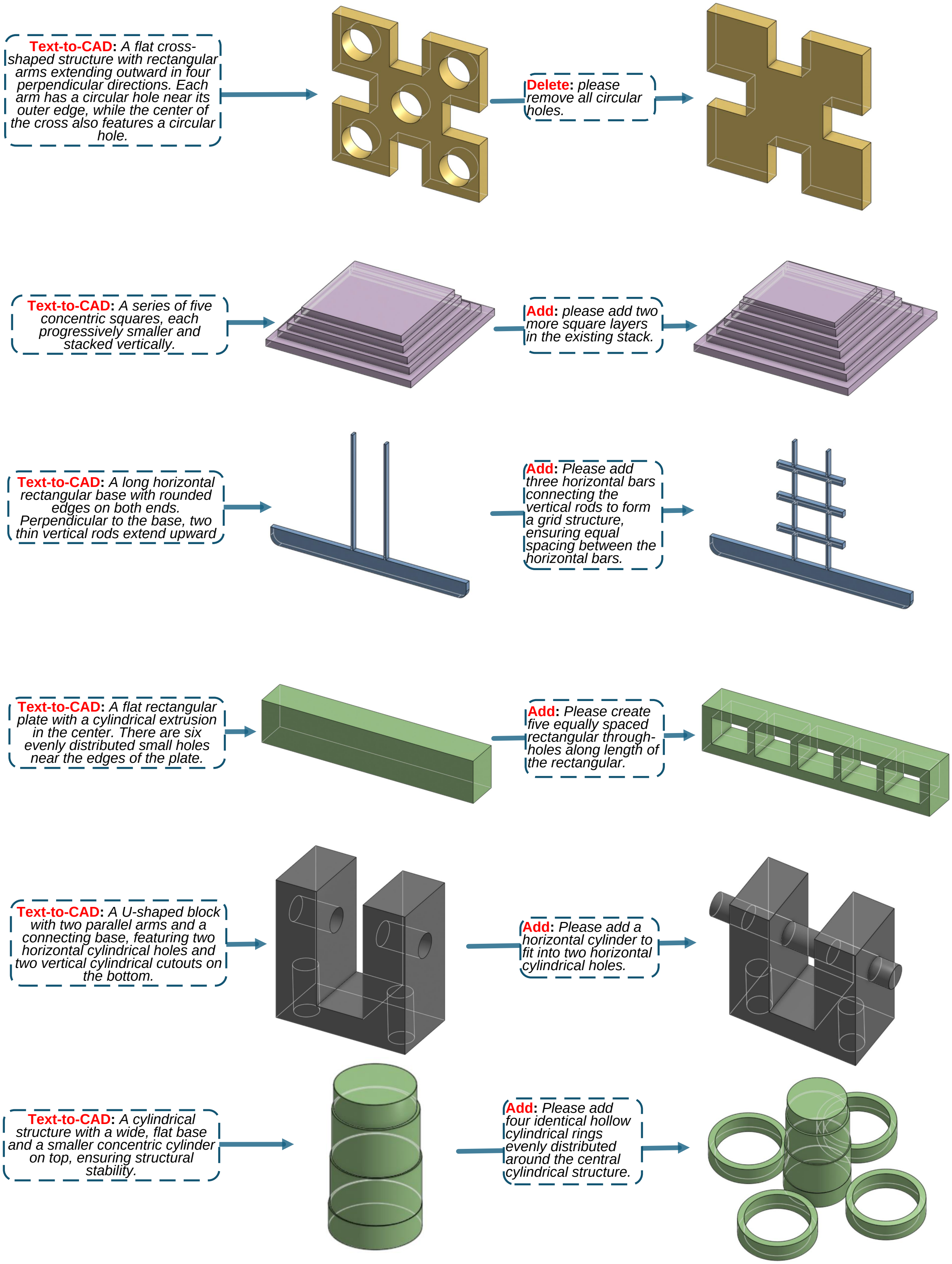


Figure 16. Supplementary working examples of Text-to-CAD, Delete, Add tasks using CAD-Llama-INS.

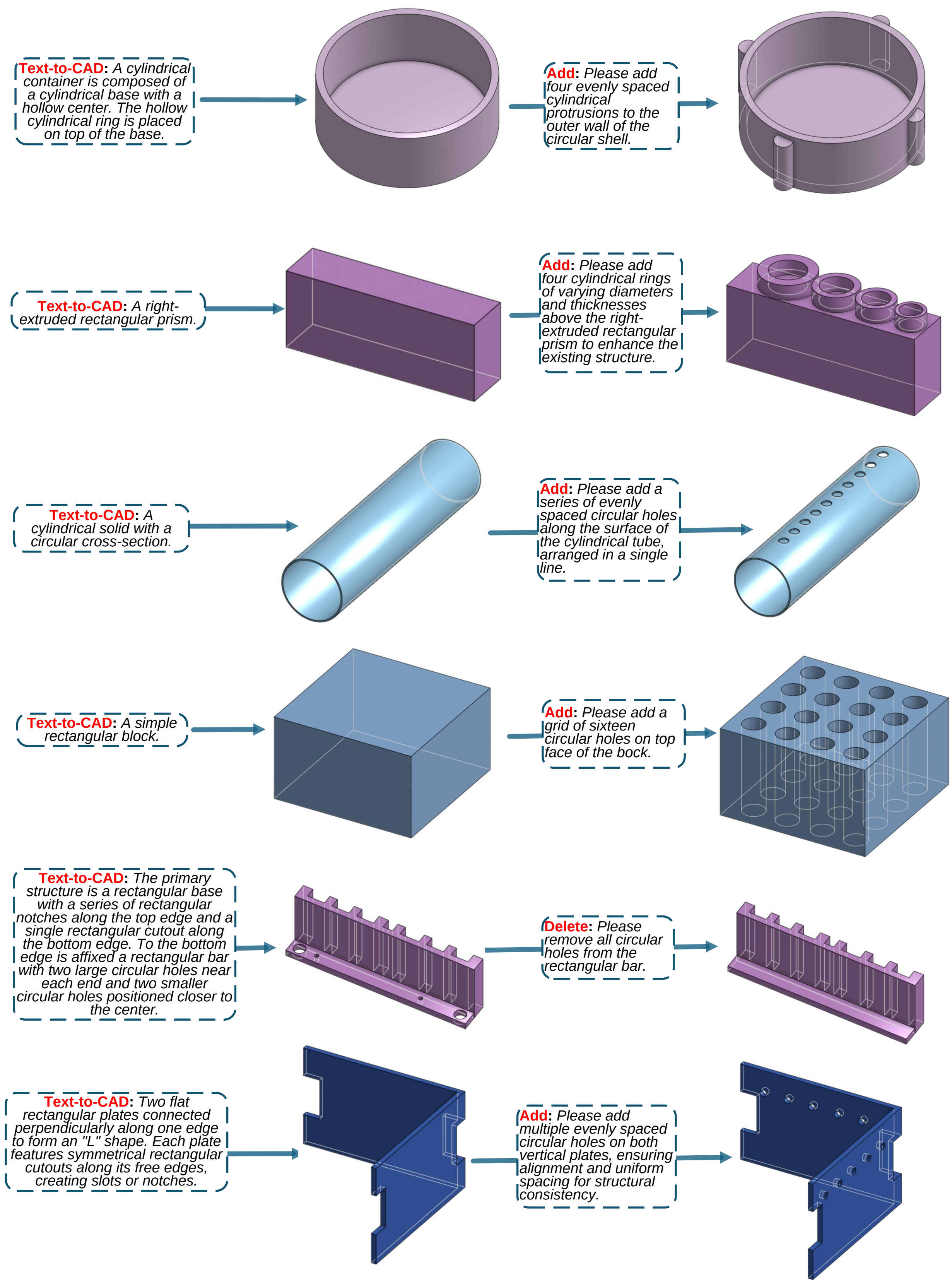


Figure 17. Supplementary working examples of Text-to-CAD, Delete, Add tasks using CAD-Llama-INS.

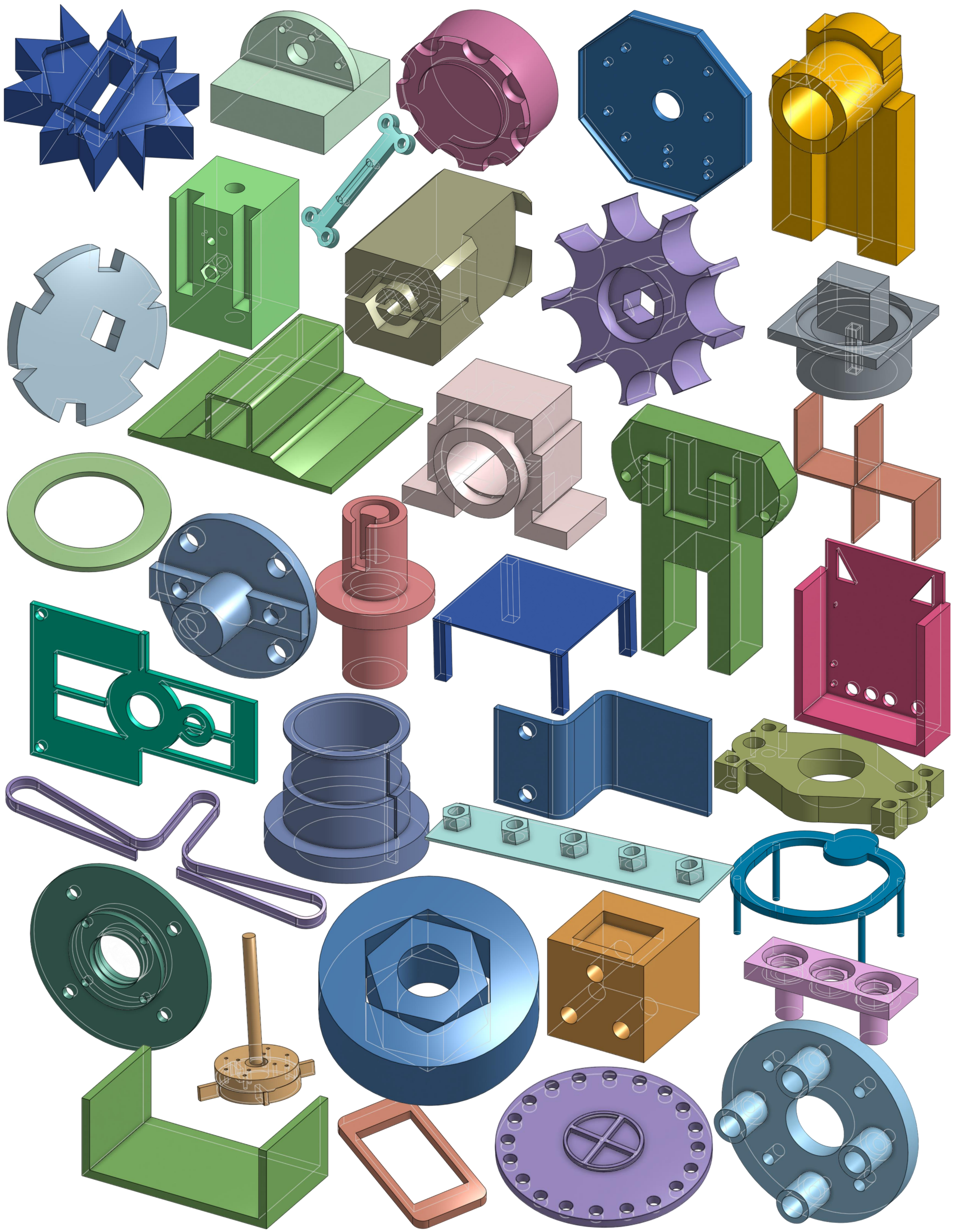


Figure 18. Supplementary results of unconditional generation produced by CAD-Llama.

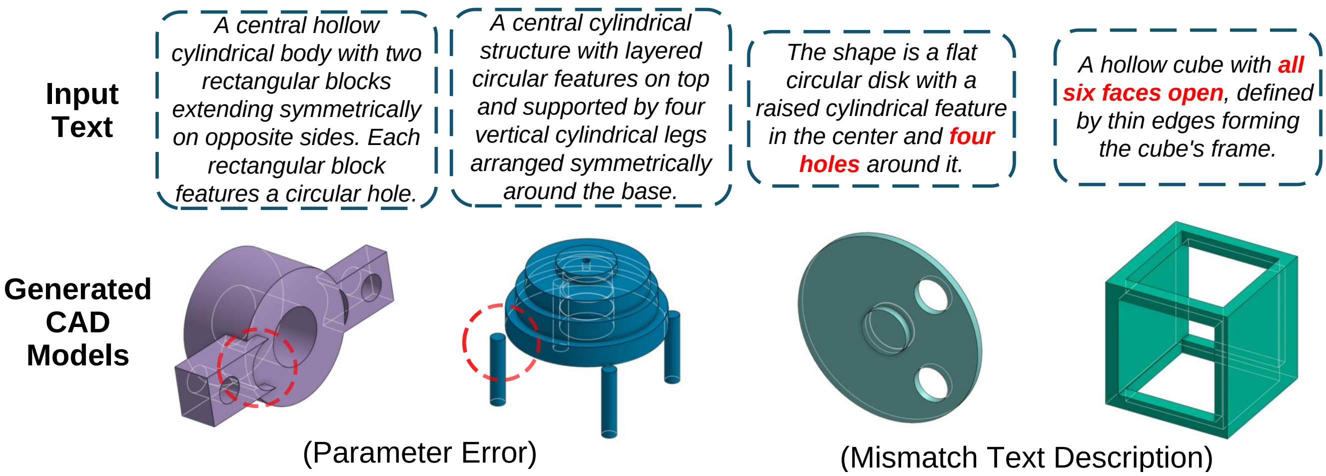


Figure 19. Failure cases for CAD-Llama-INS. We illustrate two types of errors: inaccuracies in parameter settings and misalignment with the text prompts.

References

[1] Opencascade. https://www.opencascade.com/. Accessed: 20-Oct-2023. 2

[2] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 3

[3] Weijia Shi, Sewon Min, Maria Lomeli, Chunting Zhou, Margaret Li, Gergely Szilvasy, Rich James, Xi Victoria Lin, Noah A Smith, Luke Zettlemoyer, et al. In-context pretraining: Language modeling beyond document boundaries. *arXiv preprint arXiv:2310.10638*, 2023. 3

[4] Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4):1–24, 2021. 3

[5] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6772–6782, 2021. 1, 2, 3