

ArtLLM: Generating Articulated Assets via 3D LLM

Supplementary Material

1. Implementation Details

Our ArtLLM training is built upon the codebases of LLaMA-Factory [7] and SpatialLM [3]. In each stage, the cross-entropy loss is used as the optimization objective for SFT. The multi-task data mixing ratio is set to 3:2:5. We employ a cosine learning rate scheduler with a maximum rate of $1e-5$ and a warmup ratio of 0.03.

During training, we augment 3D input point clouds with random scaling and rotations, applied to each sample with a probability of 0.75. For random scaling, we sample a scale factor $s \in [0.8, 1.05]$, and apply it to the object. For random rotation, we randomly select a rotation angle θ from 90, 180, 270, and rotate the object along the y axis. These transformations will also be applied on the part layout and articulations.

In Stage 1, we train on mixed data using Task 1 with 8 H2O GPUs for 50 epochs to obtain initial weights for the point encoder and projector, which takes approximately 8 hours. In Stage 2, we train the model on mixed data using all three tasks with 8 H2O GPUs for 30 epochs, requiring 15 hours.

The part generation model uses the publicly released version of XPart [5]. After generating the link geometry, we combine it with the previously predicted articulations and export the result in URDF format, enabling seamless integration into simulators for further analysis and simulation.

2. ArtLLM Template

Here we present the text prompts used for training ArtLLM in our multi-task training. Please note that our ArtLLM is targeted at specific tasks. Therefore, we keep the text prompts as concise as possible to reduce token usage while still enabling clear differentiation between multi-task.

```
# Task 1: Part Layout Prediction
Detect part boxes.
```

```
# Task 2: Kinematic Prediction
Given part boxes, detect joints.
```

```
# Task 3: End-to-End Prediction
Detect part boxes and joints.
```

Next, we provide the output format template for ArtLLM. Following SpatialLM [3], we adopt a concise yet sufficiently informative template as the LLM’s output format, which includes the part layout and the four types of joint articulations including RevoluteJoint, ContinuousJoint, ScrewJoint, and Prismatic Joint.

Part bounding box:

```
@dataclass
class BBox:
    min_x: int
    min_y: int
    min_z: int
    max_x: int
    max_y: int
    max_z: int
```

Revolute joint:

```
@dataclass
class RevoluteJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    axis_position: [int, int, int]
    rotation_limit: [int, int]
```

Continuous joint:

```
@dataclass
class ContinuousJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    axis_position: [int, int, int]
```

Screw joint:

```
@dataclass
class ScrewJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    axis_position: [int, int, int]
    translation_limit: [int, int]
```

Prismatic joint:

```
@dataclass
class PrismaticJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    translation_limit: [int, int]
```

3. Detail of Training Dataset

In this section, we further provide the detailed information of our curated dataset used for training ArtLLM.

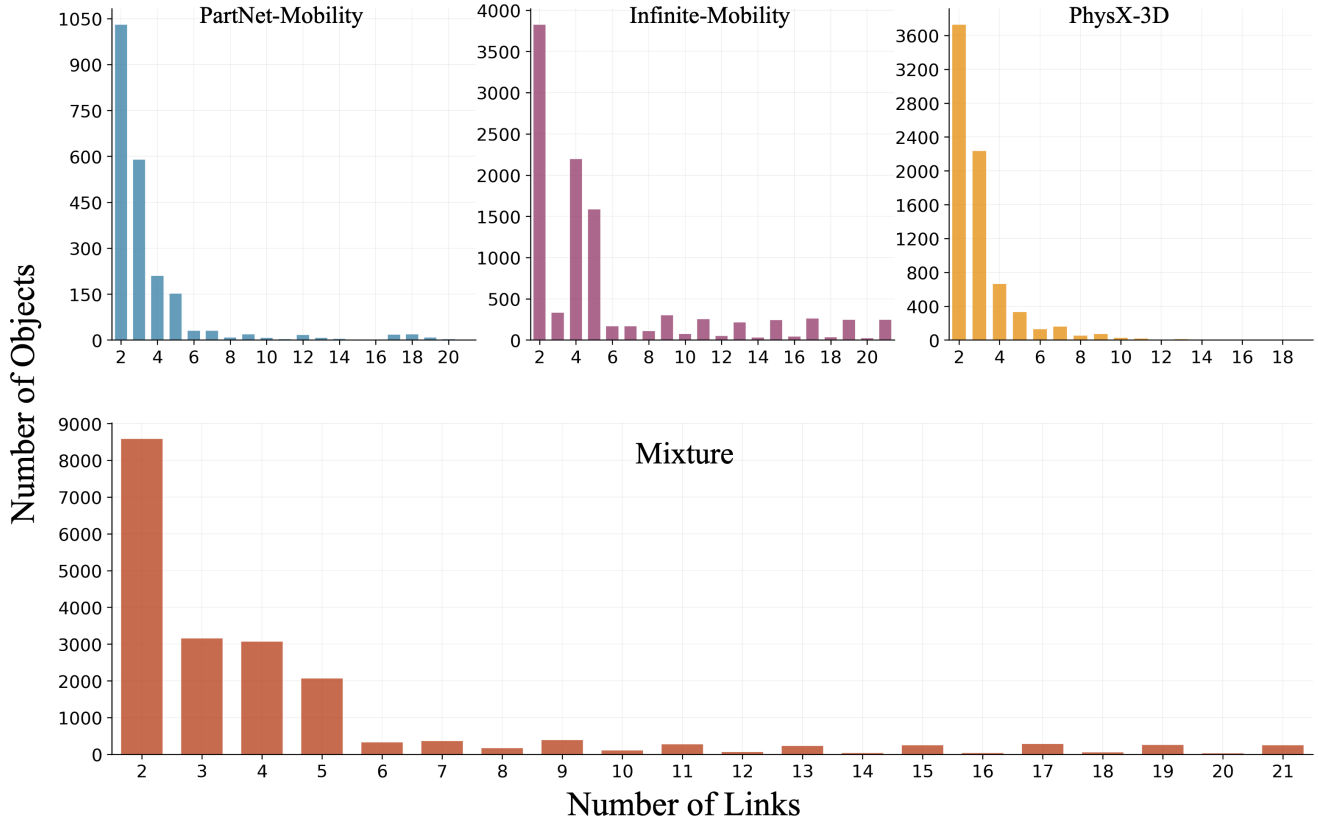


Figure 1. Part number statistic of our curated dataset for training **ArtLLM**

3.1. Detail Statistic

We begin by providing a more detailed statistical analysis of the dataset.

As shown in Figure 1, we first statistic the distribution of samples by the number of links. The curated dataset filtered sample with more than 20 joints, so the links are no more than 21. Although most objects in the dataset contain a relatively small number of parts, we have collected a sufficient number of samples with larger part counts. This ensures that our model can effectively handle objects with varying numbers of parts and generalize well across different levels of structural complexity.

3.2. Layout and articulation sort

Since the ordering of data is crucial for the training stability of autoregressive models such as LLMs, we further apply sorting operations to both the layout and articulation data to convert unordered bounding boxes and joints into ordered sequences. For the layout, following the approach used in OmniPart [6], we sort the quantized bounding boxes based on their minimum coordinates in z-y-x order. For articulation, we sort all joints according to the IDs of their child links in ascending order.

By incorporating this sorting procedure, we ensure that each sample is represented as a deterministically ordered

sequence, eliminating ambiguity in ordering and improving the stability of LLM training and inference.

3.3. Example Training Data

To facilitate a clearer understanding of the input and output formats of ArtLLM, we present a randomly selected sample from our training dataset, which is in ShareGPT format. The example includes the raw data sample with the continuous value, and the data sample with the discretized value that is actually used during training.

Raw data sample with continuous value:

```
{ "conversations": [
{
    "from": "human",
    "value":
    "<point_cloud>
Detect part boxes and joints.
The reference code is as followed:
@dataclass
class BBox:
    min_x: int
    min_y: int
    min_z: int
    max_x: int
    max_y: int
```

```

        max_z: int

@dataclass
class RevoluteJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    axis_position: [int, int, int]
    rotation_limit: [int, int]

@dataclass
class ContinuousJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    axis_position: [int, int, int]

@dataclass
class ScrewJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    axis_position: [int, int, int]
    translation_limit: [int, int]

@dataclass
class PrismaticJoint:
    parent_box_id: int
    child_box_id: int
    axis_direction: int
    translation_limit: [int, int]"
},
{
    "from": "gpt",
    "value": "<|layout_s|>
bbox_0=BBox(
    0.3900, -0.3525, 0.2381,
    0.8700, 0.2925, 0.3019
)
bbox_1=BBox(
    -0.0000, -0.3525, 0.2381,
    0.3300, 0.2925, 0.3019
)
bbox_2=BBox(
    -0.3300, -0.3525, 0.2381,
    -0.0000, 0.2925, 0.3019
)
bbox_3=BBox(
    -0.8700, -0.3525, 0.2381,
    -0.3900, 0.2925, 0.3019
)
bbox_4=BBox(
    0.3900, 0.3075, -0.1819,

```

```

    0.8700, 0.4875, 0.2953
)
bbox_5=BBox(
    -0.8700, 0.3075, -0.1819,
    -0.3900, 0.4875, 0.2953
)
bbox_6=BBox(
    -0.9000, -0.5175, -0.3019,
    0.9000, 0.5175, 0.2606
)
<|layout_e|>
<|art_s|>
joint_0=RevoluteJoint(
    6, 0, [0, 1, 0],
    [0.8672, -0.0300, 0.2400],
    [0, 90]
)
joint_1=RevoluteJoint(
    6, 1, [0, 1, 0],
    [0.3280, -0.0300, 0.2396],
    [0, 90]
)
joint_2=RevoluteJoint(
    6, 2, [0, 1, 0],
    [-0.3283, -0.0300, 0.2384],
    [0, -90]
)
joint_3=RevoluteJoint(
    6, 3, [0, 1, 0],
    [-0.8688, -0.0300, 0.2394],
    [0, -90]
)
joint_4=PrismaticJoint(
    6, 4, [0, 0, 1],
    [0.0000, 0.3717]
)
joint_5=PrismaticJoint(
    6, 5, [0, 0, 1],
    [0.0000, 0.3717]
)
<|art_e|>"
}
],
"point_clouds": ["46145/pcd/46145.ply"]
}

```

The discretized value actually used in our training. We use special tokens to represent the bounding box coordinate, axis direction, axis origin, and axis limit range.

```

<|layout_start|>
bbox_0 = BBox(
    <P_6>, <P_30>, <P_44>,
    <P_122>, <P_98>, <P_81>

```

```

)
bbox_1 = BBox(
    <P_8>, <P_83>, <P_52>,
    <P_40>, <P_96>, <P_83>
)
bbox_2 = BBox(
    <P_88>, <P_83>, <P_52>,
    <P_120>, <P_96>, <P_83>
)
bbox_3 = BBox(
    <P_8>, <P_41>, <P_79>,
    <P_40>, <P_83>, <P_84>
)
bbox_4 = BBox(
    <P_42>, <P_41>, <P_79>,
    <P_64>, <P_83>, <P_84>
)
bbox_5 = BBox(
    <P_63>, <P_41>, <P_79>,
    <P_86>, <P_83>, <P_84>
)
bbox_6 = BBox(
    <P_88>, <P_41>, <P_79>,
    <P_120>, <P_83>, <P_84>
)
<|layout_end|>
<|art_start|>
joint_0 = PrismaticJoint(
    0, 1, <D_4>,
    [<LT_32>, <LT_38>]
)
joint_1 = PrismaticJoint(
    0, 2, <D_4>,
    [<LT_32>, <LT_38>]
)
joint_2 = RevoluteJoint(
    0, 3, <D_2>,
    [<P_8>, <P_62>, <P_79>],
    [<LR_18>, <LR_24>]
)
joint_3 = RevoluteJoint(
    0, 4, <D_2>,
    [<P_42>, <P_62>, <P_79>],
    [<LR_18>, <LR_24>]
)
joint_4 = RevoluteJoint(
    0, 5, <D_2>,
    [<P_84>, <P_62>, <P_79>],
    [<LR_24>, <LR_30>]
)
joint_5 = RevoluteJoint(
    0, 6, <D_2>,
    [<P_119>, <P_62>, <P_79>],

```

```

    [<LR_24>, <LR_30>]
)
<|art_end|>

```

During training, the point cloud is replaced to point tokens produced by the point cloud encoder.

4. Detail of Experiments

4.1. Compute Resources

In our comparison experiments, all methods were evaluated using the same computation resources. Specifically, model inference was conducted on an Ubuntu server equipped with 2 * Intel Xeon Silver 4210 processor and a single NVIDIA GeForce RTX 3090 GPU with 24GB VRAM. Our model training, including ablation experiments, are conducted on a server with 2 Intel Xeon Platinum 8476C processor, and 8 * NVIDIA H20 96GB GPU with NVLink.

4.2. Metrics Compute

Below we provide the detailed definitions of the evaluation metrics used in our experiments. After applying the scale and coordinate alignment, and use Hungarian matching algorithm to align parts and joints with ground-truth instances, we compute metrics for each component as follows:

For **Part Layout**, we use the mIoU as the evaluation metric, with 3d IoU defined as:

$$\text{IoU} = \frac{V_{\text{inter}}}{V_{\text{union}}} = \frac{V_{\text{inter}}}{V_p + V_g - V_{\text{inter}}}, \quad (1)$$

and mIoU defined as

$$\text{mIoU} = \frac{1}{N} \sum_{i=1}^N \text{IoU}_i \quad (2)$$

For **articulation**, the metrics including joint type accuracy, joint axis error, joint pivot error, and joint range IoU. For **joint type**, the evaluation metric is defined as:

$$\text{acc}_{\text{type}} = \begin{cases} 1, & \text{if } \text{type}(J_p) == \text{type}(J_g) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

For **joint axis error**, we measure the angle between the predicted axis direction and the ground-truth direction. Since directions are equivalent up to sign, we compute the angle for both the original and reversed directions and take the smaller value as metric:

$$e_{\text{axis}} = \min \left(\arccos \left(\frac{a_p \cdot a_g}{\|a_p\|_2 \|a_g\|_2} \right), \arccos \left(\frac{-a_p \cdot a_g}{\|a_p\|_2 \|a_g\|_2} \right) \right), \quad (4)$$

Table 1. **Quantitative Comparison.** We show the metrics on each category of the 3 method against our method. The ‘-’ denotes that no limit is predicted due to the error joint type.

Category	Method	mIoU	Type Acc	Joint-Axis-Err	Joint-Pivot-Err	Range-IoU	Graph Acc
Table	Ours	0.4528	0.7000	0.1125	0.1393	0.6451	0.7000
	ArtAnything	0.2803	0.7500	0.2992	0.1458	0.8029	0.2500
	Singapo	0.4716	1.0000	0.0017	0.1992	0.4906	0.4000
	URDFormer	0.0999	0.6000	1.0472	0.8528	0.5941	0.1000
StorageFurniture	Ours	0.7759	0.9091	0.1771	0.1192	0.7354	0.9091
	ArtAnything	0.4464	0.9200	0.2537	0.4336	0.8555	0.8000
	Singapo	0.4741	0.8545	0.2140	0.1760	0.6092	0.5455
	URDFormer	0.0948	0.6481	0.6973	0.7580	0.8302	0.2037
WashingMachine	Ours	0.7978	1.0000	0.0493	0.0281	0.9326	1.0000
	ArtAnything	0.3889	1.0000	0.0000	1.0089	0.9191	0.0000
	Singapo	0.3364	0.0000	1.2821	0.4992	–	0.0000
	URDFormer	0.0000	0.0000	0.9329	0.2462	–	0.0000
Dishwasher	Ours	0.6359	1.0000	0.2125	0.0965	0.8333	1.0000
	ArtAnything	0.4761	1.0000	1.0472	0.0968	1.0000	1.0000
	Singapo	0.4600	1.0000	0.0039	0.5999	0.6563	1.0000
	URDFormer	0.1718	1.0000	0.5236	0.4210	0.9995	0.0000
Refrigerator	Ours	0.5916	0.7500	0.1703	0.0948	1.0000	0.7500
	ArtAnything	0.3391	0.7500	0.3927	0.2450	0.5000	0.7500
	Singapo	0.2890	1.0000	0.0059	0.0301	0.5835	0.2500
	URDFormer	0.1044	0.5000	0.7854	1.0901	0.4997	0.2500
Microwave	Ours	0.9172	1.0000	0.0000	0.0710	0.5000	1.0000
	ArtAnything	0.2327	1.0000	0.0000	1.4211	1.0000	1.0000
	Singapo	0.5561	1.0000	0.0038	0.0718	0.6768	1.0000
	URDFormer	0.1508	1.0000	0.0000	0.4621	0.9995	0.0000
Oven	Ours	0.6474	1.0000	0.1682	0.0116	0.7721	1.0000
	ArtAnything	0.2038	0.5000	1.1781	0.4022	0.9798	0.5000
	Singapo	0.4441	0.5000	0.2002	0.2205	0.6628	0.0000
	URDFormer	0.2363	0.5000	1.1781	0.4367	0.9995	0.0000

For **axis pivot error**, this evaluates the positional discrepancy between the predicted and ground-truth joint origin:

$$e_{\text{origin-pos}} = \frac{|\mathbf{p} \cdot (\mathbf{a}_p \times \mathbf{a}_g)|}{|\mathbf{a}_p \times \mathbf{a}_g|}. \quad (5)$$

where $\mathbf{p} = x_p - x_g$.

For **joint range IoU**, we similarly consider direction reversal. So we compute the error with respect to both the original and reversed directions and use the maximum one as the final metric:

$$\text{IoU}_{\text{lim}} = \max(\text{IoU}(r_p, r_g), \text{IoU}(-r_p, r_g)). \quad (6)$$

For the hierarchical structure, we evaluate the **graph accuracy** by checking whether the predicted graph is isomor-

phic to the ground-truth graph, using lib networkx:

$$acc_{\text{graph}} = \begin{cases} 1, & \text{if } \text{graph}(p) == \text{graph}(g) \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

For all metrics, we first compute the metric for each individual object, then average across all objects within the same category, and finally average over all categories to obtain the final evaluation results.

4.3. Comparison on each category

To provide a clearer comparison across categories, we include a detailed table presenting the quantitative results for each category. As shown in Tab. 1, our method demonstrates clear superiority across most metrics and categories. Although Articulate Anything [1] performs better on the

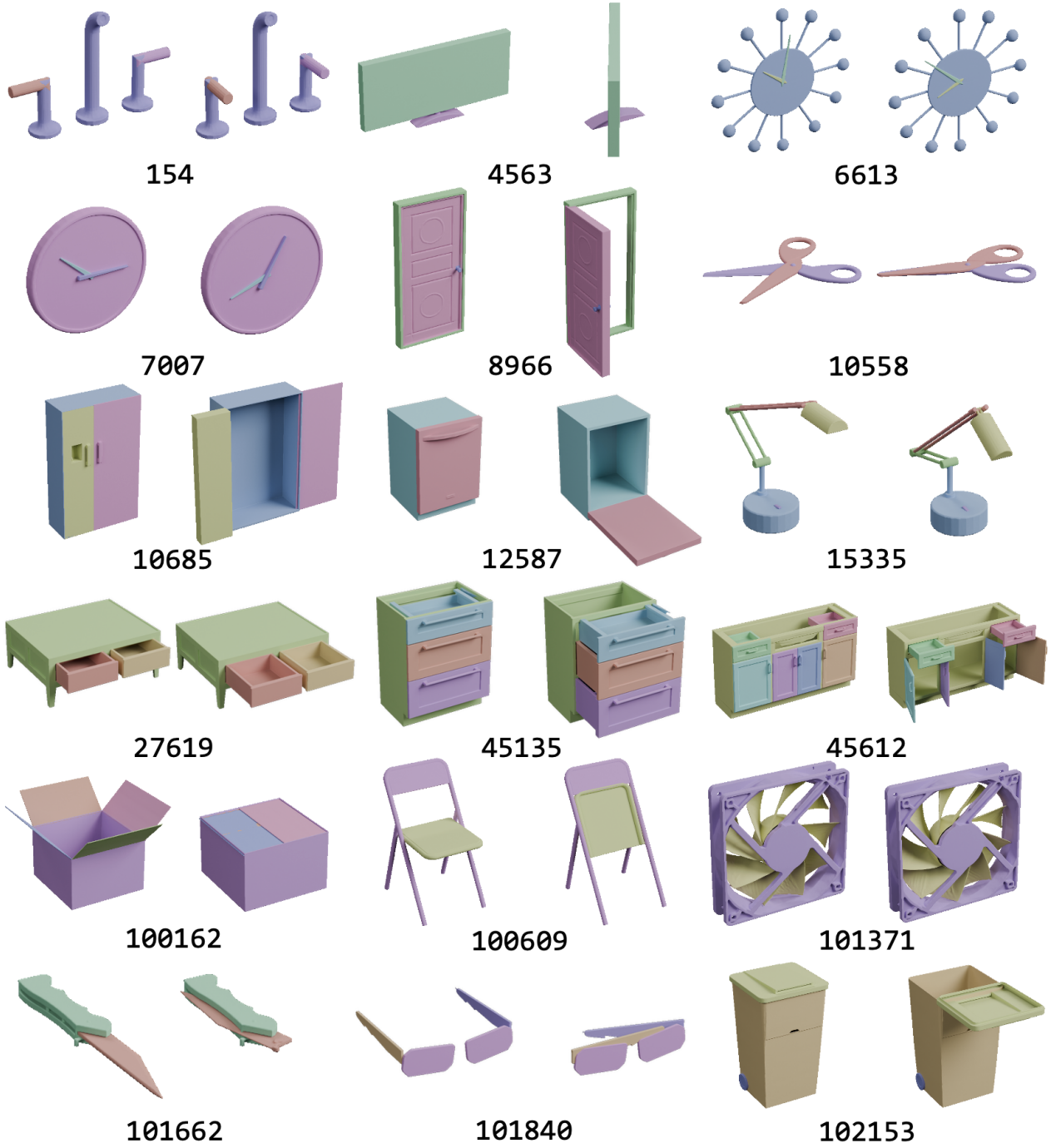


Figure 2. Additional results generated by our **ArtLLM**, from the test set of the PartNet-Mobility [4] dataset. The input geometry is generated by Hunyuan3D 3.0 from rendered images. In each pair of images, the left one is the canoical state, and the right one is a sampled articulated state.

limit range metric, it exhibits poor performance in part layout and joint origin prediction. SINGAPO [2], on the other hand, occasionally shows advantages in joint axis prediction, but its performance is significantly weaker on the re-

maining metrics, particularly on joint origin.



Figure 3. Failure cases analysis

5. More Results

Here we show the results generated by our ArtLLM on more categories in the PartNet-Mobility [4] test set, the input point cloud for our model is still generated by Hunyuan3D 3.0. As shown in Fig. 2, our model can generate realistic articulation assets from images across many categories with accurate geometry.

6. Failure Cases Analysis

Here, we also show some failure cases on the test set of PartNet-Mobility [4] dataset.

As shown in Fig. 3, in the case of 7130 and 22241, although our ArtLLM produces accurate layouts, XPart [5] fails to generate the internal concave structures when creating geometry, resulting in assets that lack realism. In Case 103303, for two parts with a high degree of overlap, the generated geometries tend to intersect with each other. In Case 103555, the detailed bread-rack structure is located inside the object and cannot be fully captured from a single image, causing it to be omitted during full-object generation. As a result, the articulation assets also fail to reproduce this internal fine structure, reducing realism.

To address these issues, future work may include fine-tuning XPart or training a 3D generative model capable of reconstructing internal object structures that are occluded in images.

References

- [1] Long Le, Jason Xie, William Liang, Hung-Ju Wang, Yue Yang, Yecheng Jason Ma, Kyle Vedder, Arjun Krishna, Dinesh Jayaraman, and Eric Eaton. Articulate-anything: Automatic modeling of articulated objects via a vision-language foundation model. *arXiv preprint arXiv:2410.13882*, 2024. 5
- [2] Jiayi Liu, Denys Iliash, Angel X Chang, Manolis Savva, and Ali Mahdavi-Amiri. Singapo: Single image controlled generation of articulated parts in objects. *arXiv preprint arXiv:2410.16499*, 2024. 6
- [3] Yongsan Mao, Junhao Zhong, Chuan Fang, Jia Zheng, Rui Tang, Hao Zhu, Ping Tan, and Zihan Zhou. Spatiallm: Training large language models for structured indoor modeling. *arXiv preprint arXiv:2506.07491*, 2025. 1
- [4] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020. 6, 7
- [5] Xinhao Yan, Jiachen Xu, Yang Li, Changfeng Ma, Yunhan Yang, Chunshi Wang, Zibo Zhao, Zeqiang Lai, Yunfei Zhao, Zhuo Chen, et al. X-part: high fidelity and structure coherent shape decomposition. *arXiv preprint arXiv:2509.08643*, 2025. 1, 7
- [6] Yunhan Yang, Yufan Zhou, Yuan-Chen Guo, Zi-Xin Zou, Yukun Huang, Ying-Tian Liu, Hao Xu, Ding Liang, Yan-Pei Cao, and Xihui Liu. Omnipart: Part-aware 3d generation with semantic decoupling and structural cohesion. *arXiv preprint arXiv:2507.06165*, 2025. 2
- [7] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*, 2024. 1