LAM: Language Articulated Object Modelers

Yipeng Gao, Yunhao Ge, Peilin Cai, Daniel Seita, Laurent Itti
University of Southern California

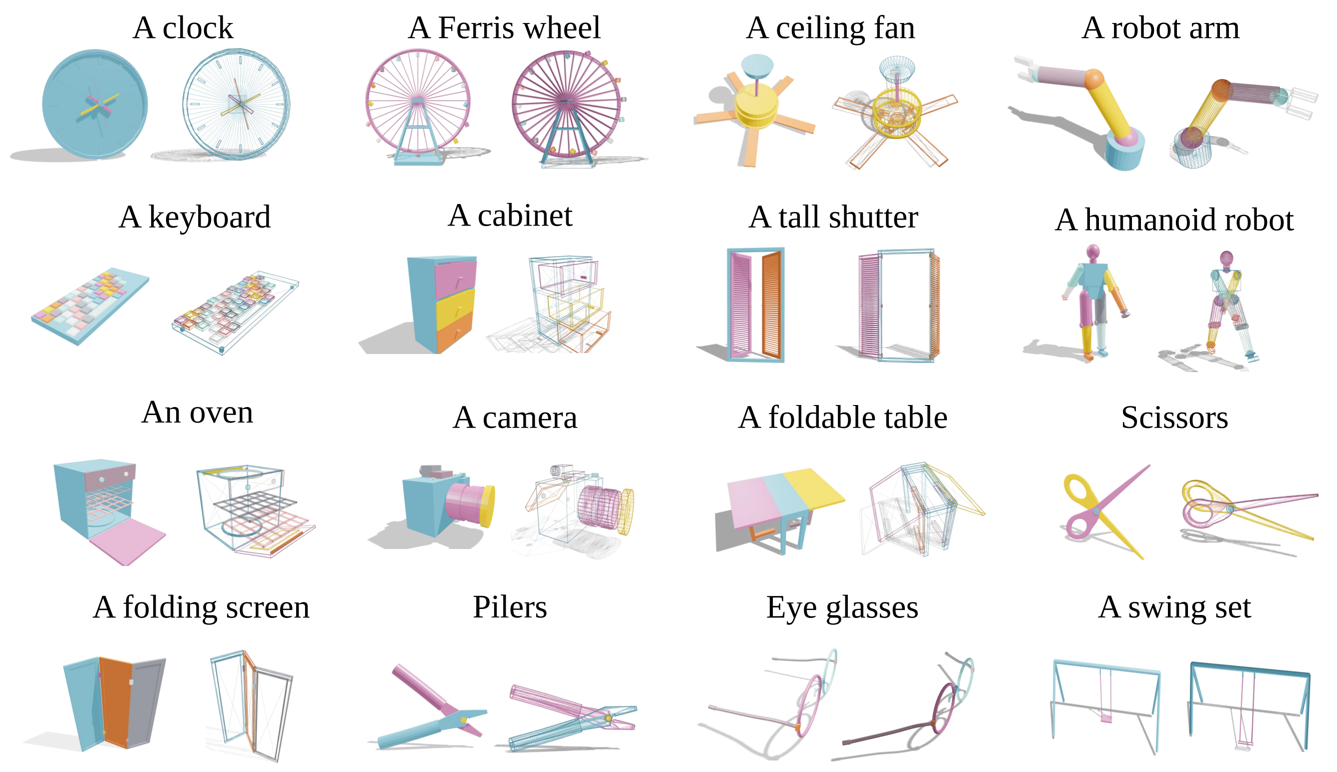

Figure 1. Our proposed LAM can generate diverse articulated objects with multiple types of movements from texts.

Abstract

We introduce LAM, a system that explores the collaboration of large-language models and vision-language models to generate articulated objects from text prompts without visual prior and pre-built 3D assets. In contrast, we formulate articulated object generation as a unified code generation task, where geometry and articulations can be co-designed from scratch. Given an input text, LAM coordinates a team of specialized modules to generate code to represent the desired articulated object procedurally. LAM first reasons about the hierarchical structure of parts (links) with Link Designer, then writes code, compiles it, and debugs it with Geometry & Articulation Coders and self-corrects with Geometry & Articulation Checkers. The code serves as a structured and interpretable bridge between individual links, ensuring correct relationships among them. Experiments demonstrate the power of leveraging code as a generative medium within a collaboration system, showcasing its effectiveness in automatically constructing complex articulated objects.

1. Introduction

Articulated objects are widespread in daily life, playing a crucial role in building realistic and interactive virtual environments for robotics, embodied AI, gaming, and VR/AR applications [7, 19, 25, 38, 42]. Despite recent progress in simulation technology that significantly accelerates training through large-scale virtual environments [33, 49], the creation of articulated 3D assets remains a critical bottleneck. Unlike static 3D objects, which are abundantly available in large open-source datasets [3, 4], articulated 3D models require expert manual annotation. This is time-consuming, as complex objects are represented as hierarchical trees of parts and sub-parts (which are called *links* in this literature), along

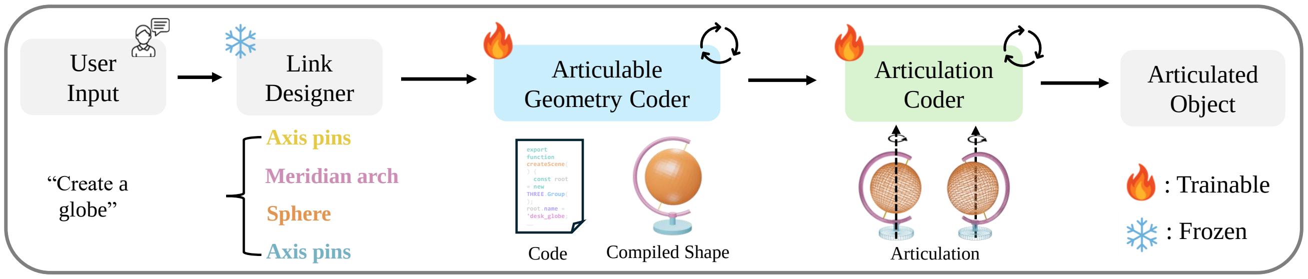

Figure 2. The overall framework of our proposed LAM. From a user's text prompt, LAM first designs a hierarchical structure of the object. It iteratively generates and refines code for both the geometry and articulation, resulting in an articulated object.

with corresponding joints, articulation types, and ranges of motion. This results in existing articulated object datasets having only thousands of instances [28, 35]. This limits the ability to leverage digital twins to train robots to interact with a broad variety of articulated objects. Automating the generation of articulation-ready models from textual descriptions represents a promising approach that we explore here to address this gap and enhance scalability in the creation of interactive virtual environments.

Previous work on articulated object modeling has primarily relied on inputs that contain structural information, such as images or videos [1, 2, 34, 44, 52], graphs [18, 26], and meshes [40, 45], to reconstruct or generate objects with movable parts, often using predefined annotations and part graphs to guide the process. However, these methods are constrained not only by their reliance on structured data as input but also by a fundamental scalability ceiling. Current training-based approaches, such as those using diffusion or graph-based architectures [18, 26], are primarily demonstrated on objects with only a limited number of parts (links) and joints. Attempting to scale these models to generate complex articulated objects (*e.g.*, a keyboard with 20 keys) is often intractable. This limitation is twofold. First, training data for such intricate, high-part-count assets is exceptionally scarce. Second, and more critically, the models themselves are not designed for this complexity; for end-to-end generation, explicitly representing the high-resolution geometry for every individual part consumes prohibitive computational resources [22], leading to out-of-memory failures or severe over-simplification as the number of links increases.

In contrast, we propose to unify the complex, coupled problem of geometry and articulation generation into a single, expressive code representation. To manage this, we first build a new dataset called **LAMBench** to bind the relation between text prompt, code and articulated objects. As shown in Figure 2, our method — *LAM* — implements a collaborative framework where a team of specialized modules (composed by LLMs and 2D&3D VLMs) work together to generate a complete, articulated 3D object from a single text prompt. This process begins with **Link Designer** that reasons about the user's text to decompose the object into a hierarchical structure from shapes to parts to links and their relationships. Following this plan, **Geometry & Articulation Coders** translate the structure into executable code for both the precise geometry of each part and their kinematic joints. That code is checked by **Debuggers** for abnormalities. A cornerstone of our system is the automated, multi-modal feedback loop, which features **Geometry & Articulation Checkers** powered by 2D and 3D Vision-Language Models (VLMs). These modules render and analyze the current object design. Then, they provide feedback, enabling the Coders to refine the code iteratively, ensuring the model is both physically plausible and visually realistic.

The key contributions of our work include: (1) We introduce *LAM*, a collaborative system where a team of specialized modules (including **Designer**, **Coders**, **Debuggers**, and **Checkers**) generates articulated objects by operating on a unified code representation for both geometry and articulation. (2) We introduce **LAMBench**, a novel dataset of text, code, and articulated object pairs, curated to enable systematic training and validation. (3) Extensive experiments validate that our method achieves superior performance for generating high-fidelity articulated objects from text.

2. Related Works

Articulated Objects Reconstruction. Early methods train end-to-end models on synthetic data, simultaneously segmenting parts and predicting joint parameters through either interaction-based [9, 11, 36, 37] or single-stage observations [8, 14, 47]. Per-object optimization techniques [24, 29] avoid training but face scalability issues with multiple joints. Real2code [34] addresses this by leveraging LLMs to generate codes for each joint. Another line of work aims to predict articulation from pre-built meshes. Articulate AnyMesh [39] and MagicArticulate [45] retrofit static meshes using VLMs and transformers, while IAAO [55] enhances reconstruction via joint affordance prediction. Recent advances employ 3D Gaussian Splatting [15]. For example, ArticulatedGS [13] builds digital twins from multi-state point clouds, RigGS [53] processes dynamic video input, and other works [16, 48, 54] integrate visual-physical modeling

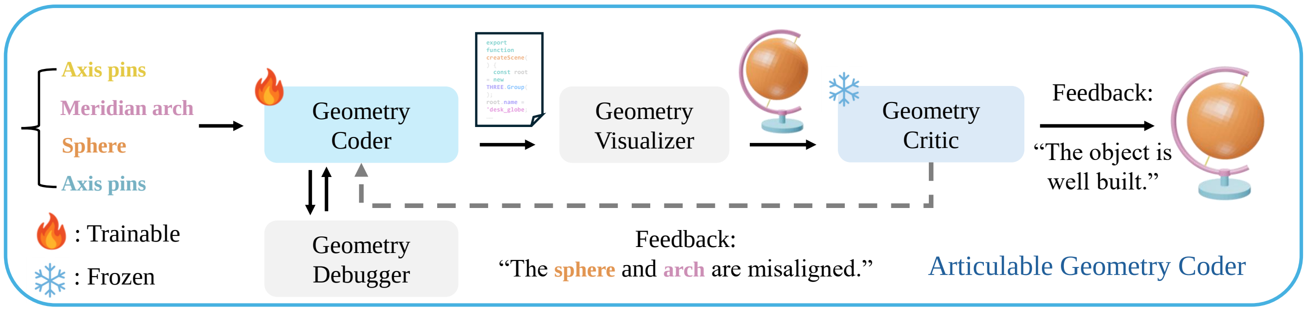

Figure 3. An overview of the Articulable Geometry Coder. Given a hierarchical link layout output by Link Designer, our Geometry Coder generates code to define the shape and position of each link. Then, a VLM-powered Geometry Checker analyzes the rendered images and provides feedback, enabling an iterative refinement loop to correct geometric errors.

with kinematic constraints.

Articulated Objects Generation. Diffusion-based methods have dominated recent advances. NAP [18] utilizes graph-attention networks. CAGE [26] and ArtFormer [46] add user controllability for specifying constraints. Single-image generation also emerged as a key direction with SINGAPO [27] learning plausible geometric variations, PhysPart [32] integrating physics constraints, and DreamArt [31] employing three-stage pipelines with diffusion priors. Meanwhile, Infinite Mobility [21] and Infinite-sim [12] scales via procedural generation. Articulate-Anything [17] synthesizes Python code compiled to URDF, Real2Code [34] reconstructs up to 10 articulated parts via LLM-based code generation, and MeshArt [6] employs hierarchical transformers for structured part-by-part generation.

In contrast, we introduce a collaborative system built upon a unified code representation that jointly models both object geometry and articulation. This integrated framework enables a closed-loop refinement process, allowing for the generation of physically plausible objects from text alone, without relying on the visual or structural priors required by previous methods.

3. LAM

3.1. Preliminaries

Representation of articulated objects. We represent articulated objects using the Unified Robot Description Format (URDF), which encodes the geometry and kinematics of object parts, called *links*. Each link $L_i = \{\mathcal{M}_i, \mathbf{T}_i\}$ consists of a 3D mesh $\mathcal{M}_i$ and a pose $\mathbf{T}_i \in \mathrm{SE}(3)$, defined by its position $\mathbf{p}_i$ and roll-pitch-yaw (RPY) orientation $\boldsymbol{\theta}_i$. A joint J_{pc} defines the kinematic connection between a parent link L_p and a child link L_c. It is formally defined as $J_{pc} = (\mathbf{T}_{pc}, t_{pc}, \mathbf{a}_{pc}, \ell_{pc})$, where $\mathbf{T}_{pc} \in \mathrm{SE}(3)$ is the joint's pose relative to the parent, t_{pc} is its type (e.g., `revolute`, `prismatic`), $\mathbf{a}_{pc} \in \mathbb{R}^3$ is the motion axis, and $\ell_{pc} = [\ell_{\min}, \ell_{\max}]$ are the motion limits. With the parent link L_p at the origin, the child link's pose $\mathbf{T}_c$ is updated by the joint motion as:

$$\mathbf{T}_c^{'} = \mathbf{T}_p \cdot \mathbf{T}_{pc} \cdot \mathbf{X}(q_{pc}) \cdot \mathbf{T}_c, \tag{1}$$

where $\mathbf{X}(q_{pc}) \in \mathrm{SE}(3)$ is the joint transformation parameterized by the motion value q_{pc} (e.g., rotation angle).

Problem Formulation. Given a textual description x, our goal is to generate an articulated object $\mathcal{A} = (\mathcal{L}, \mathcal{J})$. The object is composed of a *link set* $\mathcal{L} = \{L_i = (M_i, \mathbf{T}_i)\}_{i=1}^{N}$, containing N meshes with corresponding poses, and a *joint set* $\mathcal{J} = \left\{J_{pc} = (\mathbf{T}_{pc}, t_{pc}, \mathbf{a}_{pc}, \ell_{pc})\right\}_{(p,c)\in\mathcal{E}}$, defining the kinematic connections. A compiler Ψ converts $\mathcal{A}$ into a physically plausible URDF model $\mathcal{U} = \Psi(\mathcal{A})$.

3.2. Articulable Geometry Generation

Code-based Representation. We treat the code as a highly compressed, parametric representation of 3D geometry. Unlike explicit representations (e.g., meshes or voxels), where memory cost scales directly with geometric resolution, the cost of our code representation is largely independent of it. By using code, we can efficiently define complex objects with a large and variable number of links.

To make the structure of articulated objects tractable for LLM, we introduce a hierarchical code-based representation progressing from *shape primitives* ($\mathcal{S}$) to *parts* ($\mathcal{P}$), and finally to *links* ($\mathcal{L}$). This structured representation circumvents the control limitations of end-to-end text-to-3D methods [30, 43, 51]. We define a set of parametric primitives, $\mathcal{S} = \{\, s_k(\boldsymbol{\phi}_k)\,\}_{k=1}^{K}$, built by calling functions like `<BoxGeometry>(l,w,h)` from the Three.js library. All primitives are normalized to a shared coordinate system for consistent alignment. Given a text instruction x, the **Geometry Coder** uses these primitive functions to generate shape primitives $\{\, s_n(\boldsymbol{\phi}_n)\,\}_{n=1}^{N}$, which can be hierarchically assembled into parts and then links. The final mesh geometry $\mathcal{M}_i$ and pose $\mathbf{T}_i$ for each link are thus defined within this program.

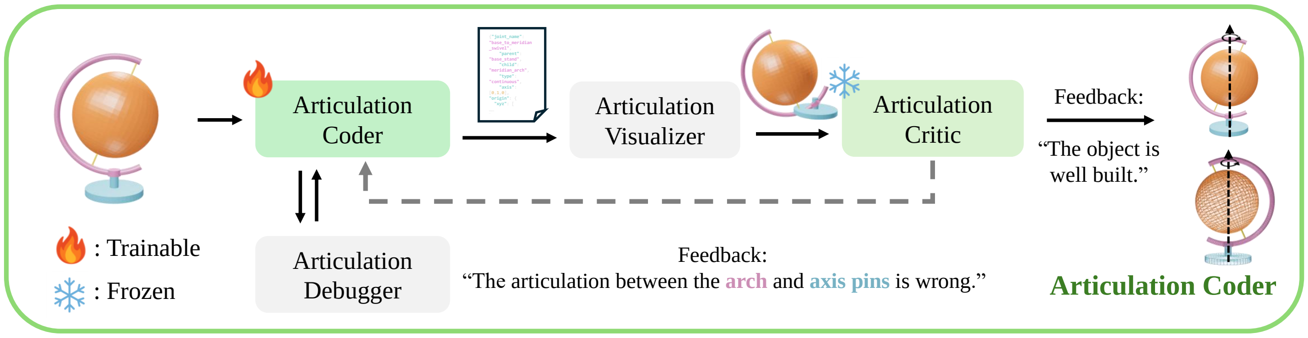

Figure 4. The Articulation Builder takes the generated shape code to define the object's articulation through a closed-loop process. An Articulation Coder generates code of joints, which the Articulation Visualizer then simulates to create a sequence of images to indicate the motion of joints. The Articulation Checker provides corrective feedback to iteratively refine the code until the motion is physically plausible and functionally correct.

Articulable Shape Generation with Iterative Refinement. As illustrated in Figure 3, given an input text, the **Link Designer** (powered by an LLM) first reasons about the prompt to decompose the target object into a hierarchical structure of links and components. The **Geometry Coder** is a large language model that translates the generated link layout into executable code by selecting and parameterizing a library of predefined functions for both shape and pose. For shape generation, it employs primitive factory functions to instantiate and compose the mesh $\mathcal{M}_i$ for each link L_i. Concurrently, it determines the appropriate pose $\mathbf{T}_i$ (including position $\mathbf{p}_i$ and orientation $\boldsymbol{\theta}_i$) for each link. This methodology offers far greater control than generating raw URDF files or using text-to-3D models, thereby mitigating issues such as over-simplification or geometric uncontrollability. Usually, the initial code may contain geometric errors or physical implausibilities due to hallucinations. Therefore, we first employ **Geometry Debugger** to automatically fix grammar issues and then develop **Geometry Checker** to correct geometric errors, which is composed of 2D VLMs (*e.g.*, GPT-4o; [10]) or 3D VLMs (*e.g.*, PointLLM; [50]). The **Geometry Visualizer** rendered multi-view images and a point cloud of the object (each link will be assigned a specific color for the Checker to refer to conveniently). Then, the **Geometry Checker** provides targeted feedback (*e.g.*, "The sphere and arch are misaligned") to enable an iterative refinement loop that corrects these errors. The final, validated link set, $\mathcal{L} = \{L_i = (M_i, \mathbf{T}_i)\}_{i=1}^N$, forms the complete object geometry $\mathcal{A}$. More details of the modules' design can refer to the supplementary material.

3.3. Articulation Generation

Once the set of links $\mathcal{L} = \{L_i = (\mathcal{M}_i, \mathbf{T}_i)\}_{i=1}^N$ is generated, the next crucial step is to define the kinematic joint set $\mathcal{J}$ that enables their articulation. This process is orchestrated by Articulation Builder, as shown in Figure 4, which interprets the geometric and semantic properties of the links to produce a functionally correct articulation structure.

Joint Assembly Solver. Our approach first simplifies the complex problem of joint placement. Since the geometry generation stage (Section 3.2) produces links that are already well-aligned within a shared world matrix system, we bypass the need to predict complex relative joint poses. Instead, we focus on predicting the essential joint parameters: the joint type t_{pc}, the parent-child link pair (L_p, L_c), and the absolute 3D position of the joint, $\mathbf{p}_{pc}$. The Articulation Builder achieves this by invoking meta-functions to analyze the spatial relationships and functional affordances of the links based on their geometry ($\mathcal{M}_i$) and pose ($\mathbf{T}_i$).

To correctly assemble the links, we designate a base link and iterate through each joint. For `prismatic` and `fixed` joints, no position update is needed as their alignment is determined during geometry generation. For `revolute` joints, we recalculate the child link's position to ensure it pivots correctly around the joint's position $\mathbf{p}_{pc}$. The updated child position $\mathbf{p}_c^{\text{new}}$ is computed as:

$$\mathbf{p}_c^{\text{new}} = \mathbf{p}_{pc} + \mathbf{R}_{pc}(\mathbf{p}_c - \mathbf{p}_{pc})$$

where $\mathbf{R}_{pc}$ is the rotation matrix derived from the joint parameters. Finally, any pose updates are recursively propagated down the kinematic chain.

Articulation Generation Using Shape Code with Checker. As illustrated in Fig. 4, the generation and validation of the joint set $\mathcal{J}$ is performed through a closed-loop, multi-agent pipeline. Taking the generated shape code as input, the **Articulation Coder**, which is a trainable large language model, generates executable code that defines the kinematic structure. It reasons about the object's components to establish parent-child hierarchies. It determines the appropriate joint type (t_{pc}), position ($\mathbf{p}_{pc}$), and motion axis ($\mathbf{a}_{pc}$) for each connection. Concurrently, a **Articulation Debugger** collaborates to resolve any syntax or code-level errors, ensuring the generated script is valid. The validated code is then passed to the **Articulation Visualizer**. To enable the

Articulation Checker to provide targeted feedback, the **Articulation Visualizer** assigns a unique color to the child link of each joint. The corresponding mapping between colors and link semantics is then passed to the 2D VLM-powered **Articulation Checker**. It assesses the functional plausibility of the object's movement. For instance, it can detect if a cabinet door opens in the wrong direction or if a drawer's movement is unnatural (as shown in Figure 4). Based on its assessment, it provides feedback (*e.g.*, "The articulation between the arch and axis pins is wrong"). This feedback guides the **Articulation Coder** to refine the code iteratively. This loop continues until the critic confirms that the articulations are well-defined and physically correct, resulting in the final, validated joint set $\mathcal{J}$. More details of the articulation generation are provided in the supplementary materials.

4. Experiment

Datasets. To ensure a fair comparison with prior works [27, 46], we conduct evaluations on the same subsets of the Part-Mobility dataset as the prior papers (5 classes for [34]; 6 classes for [46]). Furthermore, to provide a more comprehensive analysis of our method's capabilities in generating diverse articulated objects, we also extend our experiments to include all 46 object categories available in the Part-Mobility dataset, referred to as *General Classes*. For each category, we use the official rendered images to generate one caption per category. Meanwhile, we also utilize the proposed **LAMBench**, which consists of 2k text, code, and articulated objects pairs for fine-tuning open-sourced large language models to generate code. Moreover, we use a more challenging set of 27 descriptions of complex articulated objects from **LAMBench**, noted as *Open-World Classes*. Please refer to the supplementary material for more details.

Benchmark and Metrics. We first adopt a masked URDF reconstruction task to validate joint placement ability and evaluate the success rate as defined in work [17]. We also measure geometric quality and diversity using **Minimum Matching Distance (MMD)**, **Coverage (COV)**, and **1-Nearest Neighbor Accuracy (1-NNA)** [26, 46]. Text-to-image alignment is quantified via CLIP [41] and BLIP [20] scores. For automated evaluation, GPT-5 [23] performs articulation examinations and pairwise preference comparisons. Finally, we use the accuracy of the generated articulated objects (both the links and the articulations should be correct) of the collected 83 captions to ablate the variant designs of LAM.

Implementation Details. Our framework centrally employs LLMs and VLMs for generating the code that defines object geometry and articulation. The Linker Designer is implemented for GPT-4o by default. For the Articulable Geometry Generation, we use Gemini-2.5-pro and functions defined from the Three.js library by default. We use o3 equipped with the proposed Joint Assembly Solver as Articulation Coder. Geometry & Articulation Checkers are based on the Gemini-2.5-flash and PointLLM [50]. The Debuggers are also Gemini-2.5-flash with deterministic Python & JavaScript scripts to verify the issues. The above setting is noted as *zero-shot*. We also test the zero-shot ability and the potential of open-sourced large language model fine-tuned by our proposed LAMBench, such as Llama 3.3 70B [5]. Please refer to the supplementary material for more details.

Table 1. Quantitative comparisons on the success rate of text-based joint prediction. Results are reported on (a) the 5 categories used by Real2Code (`Laptop`, `Box`, `Refrigerator`, `Storage-Furniture`, `Table`), and (b) all 40 classes of the Part-Mobility dataset. * denotes using the default commercial LLMs. Zero-shot means all the modules of LAM are used as Llama 3.3 70B and finetuned means that Llama 3.3 70B is finetuned using our LAMBench dataset.

Method	Success Rate (%)	
	Five Classes	General Classes
Real2Code	13.5	–
URDFormer	–	14.6
Articulate Anything	40.3	48.9
LAM *	**77.1**	**68.2**
LAM (zero-shot)	53.7	56.9
LAM (finetuned)	69.3	64.8

Table 2. Visual alignment (CLIP, BLIP) and articulation modeling (GPT-5 pass rate) results on the shared classes (`Storage Furniture`, `Table`, `Refrigerator`, `Dishwasher`, `Oven`, `Washer`) between CAGE and Singapo.

Method	**CLIP ↑**	**BLIP ↑**	**GPT-5 ↑**
CAGE	27.65	53.92	53.9%
SINGAPO	30.43	56.21	58.8%
Articulate Anything	28.23	56.99	65.3%
LAM*	**31.94**	**63.76**	**77.0%**
LAM (zero-shot)	30.73	59.48	69.4%
LAM (finetuned)	31.22	61.74	72.3%

4.1. Main Results

Success Rate Comparison of Joint Prediction. In Table 1, on the dataset classes from Real2Code, our default 'LAM *' model achieves a success rate of 77.1%, which significantly surpasses both Articulate Anything (40.3%) and Real2Code (13.5%). This robust performance is consistent even on the more diverse General Classes, where 'LAM *' attains a 68.2% success rate, again outperforming the strongest baseline, Articulate Anything (48.9%). Furthermore, we observe that our open-source model using Llama 3.3 70B ('LAM (zero-shot)') improves significantly after instruction tuning on our new LAMBench dataset, jumping from 53.7% to 69.3% (Five Classes) and 56.9% to 64.8% (General Classes).

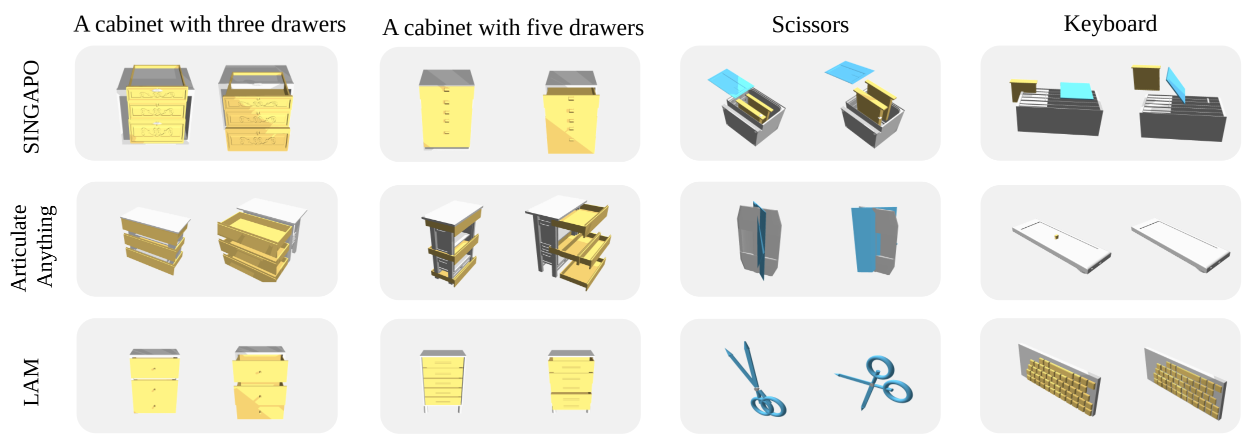

Figure 5. Generation quality comparisons across different classes. SINGAPO fails to produce sensible objects in the OOD classes. Articulate Anything also struggles with the keyboard, and scissors.

Table 3. In-distribution generation quality comparison using MMD (lower is better), COV (higher is better), and 1-NNA (lower is better). * denotes using the default commercial LLMs. Zero-shot means all the modules of LAM are used as Llama 3.3 70B and finetuned means that Llama 3.3 70B is finetuned using our LAMBench dataset.

Method	**MMD** ↓	**COV** ↑	**1-NNA** ↓
CAGE	0.0193	0.6064	0.5319
ArtFormer	0.0292	0.5213	0.5266
ArtFormer-PR	0.0214	0.6400	0.3950
LAM*	**0.0149**	0.6871	**0.3599**
LAM (zero-shot)	0.0179	0.6582	0.3762
LAM (finetuned)	0.0162	**0.6873**	0.3697

These experiments validate the superior capability of our method and the effectiveness of our proposed LAMBench dataset.

Visual Alignment and Generation Quality Comparisons. Table 2 and Table 3 provide a comprehensive evaluation of our LAM model against several baselines, assessing both the visual-semantic alignment with text prompts and the quality of in-distribution generation. In the visual alignment and articulation preference comparisons, our default LAM* model demonstrates clear superiority. It achieves the highest CLIP and BLIP scores (31.94 and 63.76, respectively), indicating a stronger semantic correspondence compared to CAGE, SINGAPO, and Articulate Anything. Furthermore, our LAM* model achieves a GPT-5 pass rate of 77.0%, indicating that its generated articulations are overwhelmingly considered functionally correct and plausible. Notably, our open-source Llama 3.3 70B model, when finetuned on LAMBench (‘LAM (finetuned)‘), also achieves strong results (31.22 CLIP, 72.3% GPT-5 pass rate), showing significant improvement over its zero-shot counterpart and validating the efficacy of our dataset. For in-distribution generation quality, our approach continues to excel, achieving the best performance across all standard metrics. It records the lowest MMD (0.0149) and 1-NNA (0.3599), which confirms that the distribution of our generated shapes is closer to the ground-truth data and more realistic. Concurrently, LAM scores the highest in COV (0.6871), reflecting its capability to produce a more diverse set of objects that better covers the data manifold.

Comparisons on General Classes. As shown in Figure 6, our LAM model demonstrates substantially better performance than Articulate Anything on both General and the more challenging Open-World object classes. For General Classes, LAM achieves significantly higher visual-semantic alignment with CLIP and BLIP scores of 31.21 and 58.94, respectively, compared to the baseline’s 25.34 and 48.32. More importantly, it garners an overwhelming preference from both GPT-4o (81.1%) and human users (84.6%). These strong preference rates from both automated and human evaluators underscore that the objects generated by LAM are not only semantically aligned but also perceived as more functionally plausible and visually coherent. This performance gap widens in the Open-World evaluation, where LAM’s user preference score reaches 91.7%, showcasing its superior generalization and ability to generate plausible articulated objects from diverse, unseen text prompts.

4.2. Ablation Studies

We utilize the combination of captions from General Classes from the Part-Mobility dataset and self-collected descriptions of Open-World Classes to evaluate the performance of different settings, resulting in a total of 83 classes. For each category, I generate one object per class for validation. We use accuracy (Acc.) to judge each setting, which

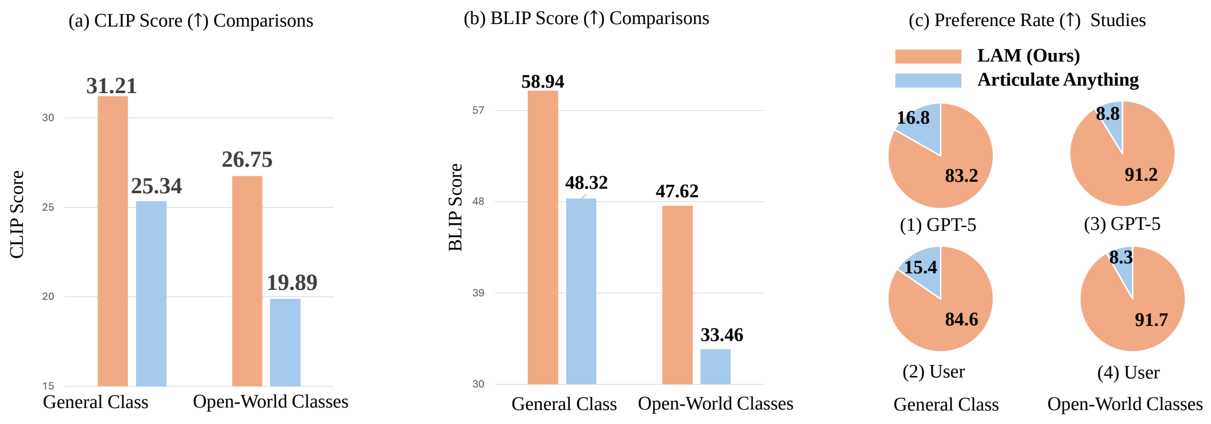

Figure 6. System-level comparisons for General and Open-World classes. For open-world classes, we collect a list of text descriptions about diverse articulated objects in the world, such as `Ferris wheel`, `shutter`, etc. (a) LAM achieves the best CLIP score on both General Classes and the new Open-World Classes. (b) LAM also achieves the best BLIP scores. (c) Both GPT-4o and human participants in our user study prefer the objects (given simulated videos to show motion) generated by LAM over those generated by Articulate Anything.

Table 4. Ablation study on the effect of Geometry Checker and Articulation Checker. Multi-view refers to using four rendered images instead of a single image for the Geometry Checker.

Geometry Checker	Articulation Checker	Max Iter	Acc. ↑
✗	✗	-	50.6%
✓	✗	-	61.4%
✗	✓	1	56.6%
✓	✓	1	66.3%
✓	✓	3	**75.9%**

Table 5. Effects of different Checker designs. Image Sequence refers to using multiple intermediate motion states for the Articulation Checker.

Geometry Checker Type	Multi-View	Images Sequence	Acc. ↑
2D	✗	✗	60.2%
2D	✓	✗	65.1%
2D	✓	✓	71.1%
3D	✓	✓	62.7%
2D & 3D	✓	✓	**75.9%**

Table 6. Ablation study on the choice of LLMs for the Planner, Shape Coder, and Articulation Coder.

Link Designer	Shape Coder	Articulation Coder	Acc. ↑
GPT-4o	GPT-4o	GPT-4o	53.8%
Gemini-2.5-flash	Gemini-2.5-flash	o3	57.9%
o3	Gemini-2.5-pro	GPT-4o	66.3%
GPT-5	Gemini-2.5-pro	GPT-4o	67.5%
GPT-4o	GPT-5	o3	64.3%
GPT-4o	Gemini-2.5-pro	GPT-4o	69.8%
GPT-4o	Gemini-2.5-pro	o3	75.9%
GPT-4o	Claude-4.1-Opus	o3	**80.7%**

means the generated objects should at least include the correct shape layout and joints with accurate placements. To evaluate the potential of our framework, our default setting utilizes commercial large language models as described in the implementation details.

Effects of Checkers. As shown in Table 4, our proposed Geometry & Articulation Checkers are vital. The baseline accuracy without any Checker is 50.6%. Introducing the Geometry Checker or Articulation Checker alone improves accuracy to 61.4% and 56.6%, respectively. Employing them together raises the accuracy to 66.3%, indicating their complementary roles. Increasing the refinement iterations to three achieves the highest accuracy of 75.9%, which highlights the effectiveness of the iterative feedback loop in generating plausible objects.

Effects of the design of Checkers. Table 5 shows the impact of Checker design choices. A basic 2D Checker using a single image yields 60.2% accuracy. This increases to 65.1% when using multi-view images and further to 71.1% with the addition of image sequences to evaluate motion. While a 3D-only Checker is less effective (62.7%), a hybrid approach combining both 2D and 3D Checkers achieves the best performance at 75.9%. This suggests that 2D and 3D Checkers provide complementary feedback, making their combination the most effective configuration.

Effects of the choice of LLMs. We ablate the core LLM components in Table 6. The results show that the model choice is critical, particularly for the **Link Designer** and **Shape Coder** roles. A strong link designer like GPT-4o is essential for high-level structure, as using weaker planners

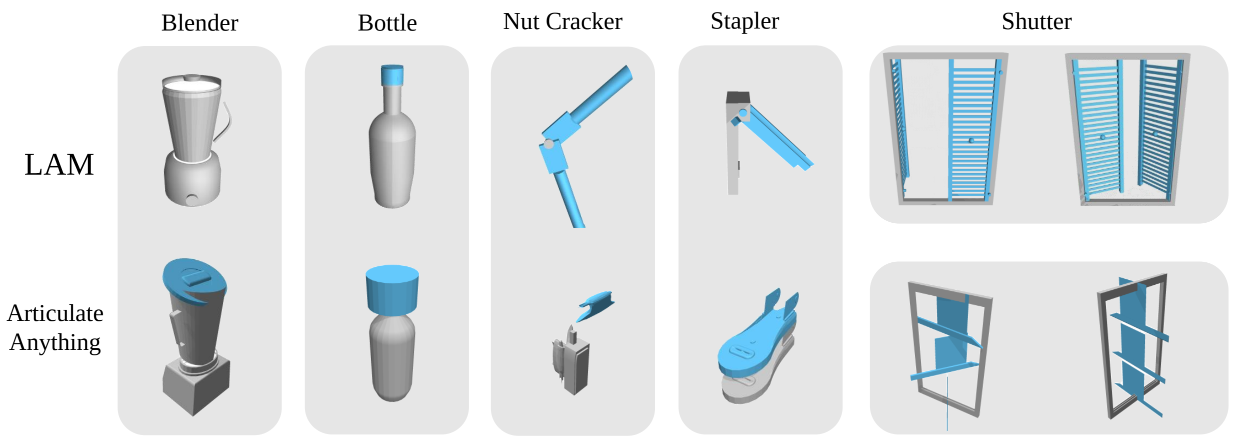

Figure 7. **Open-Vocabulary Scenarios.** Our model consistently outperforms Articulate Anything.

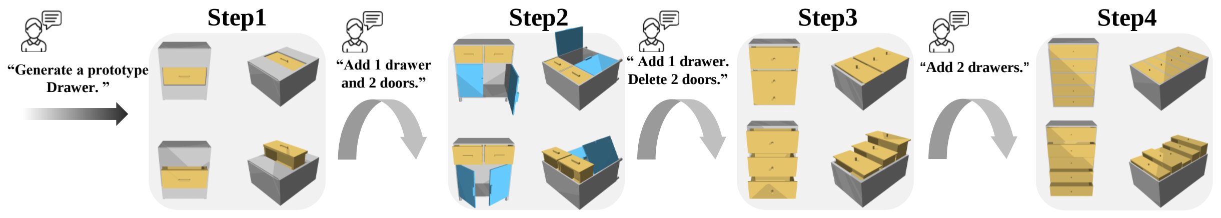
"Generate a prototype Drawer."
Step1
"Add 1 drawer and 2 doors."
Step2
"Add 1 drawer. Delete 2 doors."
Step3
"Add 2 drawers."
Step4

Figure 8. **Instruction-following ability.** In four steps, including adding and removing sub-objects, a one-drawer cabinet is guided to become a five-drawer cabinet.

significantly degrades performance. The greatest sensitivity is in the **Shape Coder**, where advanced models like Claude-4.1-Opus (80.7% Acc.) and Gemini-2.5-pro (75.9% Acc.) dramatically outperform other combinations. Interestingly, the specialized o3 model proves most effective for the **Articulation Coder** task, suggesting it benefits from a specialized model over a general-purpose one.

4.3. Qualitative Results

Overall qualitative comparisons. Figure 5 illustrates our method's performance across six diverse zero-shot targets: simple (3- and 5-drawer cabinets) and OOD (keyboard, scissors). Our pipeline successfully encodes each link as a precisely posed URDF mesh and accurately predicts all joints. The output is always collision-free and correctly articulated, whereas Singapo and Articulate Anything frequently misplace parts or omit hinges and keys. The combination of stability on simpler tasks, excellent visual quality on more challenging ones, and strong generalization to OOD examples clearly demonstrates the superiority of our approach.

Open-Vocabulary Scenarios. Figure 7 compares our model with Articulate Anything across three domains—containers (spatial reasoning), tools (precision), and complex furniture (structural complexity). Our system shows stronger command understanding and physical common sense: it tracks part-to-part spatial relations more accurately, identifies movable or interactive components more explicitly, and handles highly intricate, mesh-like structures and dense layouts.

Instruction-following Ability. Integrating high-context LLMs into our pipeline makes the system portable and reusable, chiefly by enabling instruction following. Prior outputs can feed later stages, so the model refines its own work—cutting users' descriptive burden, supporting incremental edits of complex objects, and allowing repeated iterations. Figure 8 shows that in four steps (including adding and removing), a one-drawer cabinet can be instructed to become a five-drawer cabinet.

5. Conclusion

We introduced LAM, a pioneering system that generates articulated 3D objects from text by unifying geometry and articulation within a single code representation. Our framework uniquely employs a collaborative team of specialized AI modules—including Designers, Coders, and Checkers—to iteratively write, debug, and refine this code through a closed-loop, multi-modal feedback process. Extensive experiments demonstrate that LAM significantly surpasses previous methods in generation quality, text alignment, and diversity, particularly showcasing robust generalization on challenging open-world classes. By streamlining the creation of articulation-ready assets, LAM offers a promising solution for applications in robotics, VR/AR, and simulation.

Acknowledgment

This work was supported by the National Eye Institute (grant R61EY037527), the National Science Foundation (award 2318101), and a research grant from Sandia National Laboratories. The authors affirm that the views expressed herein are solely their own, and do not represent the views of the United States government or any agency thereof.

References

[1] Mehmet Aygun and Oisin Mac Aodha. Saor: Single-view articulated object reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10382–10391, 2024. 2

[2] Chuhao Chen, Isabella Liu, Xinyue Wei, Hao Su, and Minghua Liu. Freeart3d: Training-free articulated object generation using 3d diffusion. *arXiv preprint arXiv:2510.25765*, 2025. 2

[3] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, et al. Objaverse-xl: A universe of 10m+ 3d objects. *Advances in Neural Information Processing Systems*, 36:35799–35813, 2023. 1

[4] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023. 1

[5] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407, 2024. 5

[6] Daoyi Gao, Yawar Siddiqui, Lei Li, and Angela Dai. Meshart: Generating articulated meshes with structure-guided transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025. 3

[7] Yunhao Ge, Yihe Tang, Jiashu Xu, Cem Gokmen, Chengshu Li, Wensi Ai, Benjamin Jose Martinez, Arman Aydin, Mona Anvari, Ayush K Chakravarthy, et al. Behavior vision suite: Customizable dataset generation via simulation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22401–22412, 2024. 1

[8] Nick Heppert, Muhammad Zubair Irshad, Sergey Zakharov, Katherine Liu, Rares Andrei Ambrus, Jeannette Bohg, Abhinav Valada, and Thomas Kollar. Carto: Category and joint agnostic reconstruction of articulated objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21201–21210, 2023. 2

[9] Cheng-Chun Hsu, Zhenyu Jiang, and Yuke Zhu. Ditto in the house: Building articulation models of indoor scenes through interactive perception. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3933–3939. IEEE, 2023. 2

[10] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 4

[11] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. Ditto: Building digital twins of articulated objects from interaction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5616–5626, 2022. 2

[12] Abhishek Joshi, Beining Han, Jack Nugent, Yiming Zuo, Jonathan Liu, Hongyu Wen, Stamatis Alexandropoulos, Tao Sun, Alexander Raistrick, Gaowen Liu, et al. Infinigen-sim: Procedural generation of articulated simulation assets. *arXiv preprint arXiv:2505.10755*, 2025. 3

[13] Guo Junfu, Yu Xin, Liu Gaoyi, et al. Articulatedgs: Self-supervised digital twin modeling of articulated objects using 3d gaussian splatting. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025. 2

[14] Yuki Kawana, Yusuke Mukuta, and Tatsuya Harada. Unsupervised pose-aware part decomposition for man-made articulated objects. In *European Conference on Computer Vision*, pages 558–575. Springer, 2022. 2

[15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 2

[16] Seungyeon Kim, Junsu Ha, Young Hun Kim, Yonghyeon Lee, and Frank C Park. Screwsplat: An end-to-end method for articulated object recognition. *arXiv preprint arXiv:2508.02146*, 2025. 2

[17] 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. 3, 5

[18] Jiahui Lei, Congyue Deng, William B Shen, Leonidas J Guibas, and Kostas Daniilidis. Nap: Neural 3d articulated object prior. *Advances in Neural Information Processing Systems*, 36:31878–31894, 2023. 2, 3

[19] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023. 1

[20] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR, 2022. 5

[21] Xinyu Lian, Zichao Yu, Ruiming Liang, Yitong Wang, Li Ray Luo, Kaixu Chen, Yuanzhen Zhou, Qihong Tang, Xudong Xu, Zhaoyang Lyu, et al. Infinite mobility: Scalable high-fidelity synthesis of articulated objects via procedural generation. *arXiv preprint arXiv:2503.13424*, 2025. 3

[22] Yuchen Lin, Chenguo Lin, Panwang Pan, Honglei Yan, Yiqiang Feng, Yadong Mu, and Katerina Fragkiadaki. Partcrafter: Structured 3d mesh generation via compositional latent diffusion transformers. *arXiv preprint arXiv:2506.05573*, 2025. 2

[23] Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *European Conference on Computer Vision*, pages 366–384. Springer, 2024. 5

[24] Jiayi Liu, Ali Mahdavi-Amiri, and Manolis Savva. PARIS: Part-level reconstruction and motion analysis for articulated objects. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 352–363, 2023. 2

[25] Jiayi Liu, Manolis Savva, and Ali Mahdavi-Amiri. Survey on modeling of articulated objects. *arXiv e-prints*, pages arXiv–2403, 2024. 1

[26] Jiayi Liu, Hou In Ivan Tam, Ali Mahdavi-Amiri, and Manolis Savva. CAGE: Controllable Articulation GEneration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2, 3, 5

[27] Jiayi Liu, Denys Iliash, Angel X. Chang, Manolis Savva, and Ali Mahdavi-Amiri. SINGAPO: Single Image Controlled Generation of Articulated Parts in Objects. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. 3, 5

[28] Liu Liu, Wenqiang Xu, Haoyuan Fu, Sucheng Qian, Qiaojun Yu, Yang Han, and Cewu Lu. Akb-48: A real-world articulated object knowledge base. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14809–14818, 2022. 2

[29] Shaowei Liu, Saurabh Gupta, and Shenlong Wang. Building rearticulable models for arbitrary 3d objects from 4d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21138–21147, 2023. 2

[30] Xiaoxiao Long, Yuan-Chen Guo, Cheng Lin, Yuan Liu, Zhiyang Dou, Lingjie Liu, Yuexin Ma, Song-Hai Zhang, Marc Habermann, Christian Theobalt, et al. Wonder3d: Single image to 3d using cross-domain diffusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9970–9980, 2024. 3

[31] Ruijie Lu, Yu Liu, Jiaxiang Tang, Junfeng Ni, Yuxiang Wang, Diwen Wan, Gang Zeng, Yixin Chen, and Siyuan Huang. Dreamart: Generating interactable articulated objects from a single image. *arXiv preprint arXiv:2507.05763*, 2025. 3

[32] Rundong Luo, Haoran Geng, Congyue Deng, Puhao Li, Zan Wang, Baoxiong Jia, Leonidas Guibas, and Siyuan Huang. Physpart: Physically plausible part completion for interactable objects. *arXiv preprint arXiv:2408.13724*, 2024. 3

[33] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021. 1

[34] Zhao Mandi, Yijia Weng, Dominik Bauer, and Shuran Song. Real2code: Reconstruct articulated objects via code generation. *arXiv preprint arXiv:2406.08474*, 2024. 2, 3, 5

[35] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 909–918, 2019. 2

[36] Jiteng Mu, Weichao Qiu, Adam Kortylewski, Alan Yuille, Nuno Vasconcelos, and Xiaolong Wang. A-sdf: Learning disentangled signed distance functions for articulated shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13001–13011, 2021. 2

[37] Neil Nie, Samir Yitzhak Gadre, Kiana Ehsani, and Shuran Song. Structure from action: Learning interactions for articulated object 3d structure discovery. *arXiv preprint arXiv:2207.08997*, 2022. 2

[38] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024. 1

[39] Xiaowen Qiu, Jincheng Yang, Yian Wang, Zhehuan Chen, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. Articulate anymesh: Open-vocabulary 3d articulated objects modeling. *arXiv preprint arXiv:2502.02590*, 2025. 2

[40] Xiaowen Qiu, Jincheng Yang, Yian Wang, Zhehuan Chen, Yufei Wang, Tsun-Hsuan Wang, Zhou Xian, and Chuang Gan. Articulate anymesh: Open-vocabulary 3d articulated objects modeling. *arXiv preprint arXiv:2502.02590*, 2025. 2

[41] 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. 5

[42] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D'Arpino, Shyamal Buch, Sanjana Srivastava, Lyne Tchapmi, et al. igibson 1.0: A simulation environment for interactive tasks in large realistic scenes. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7520–7527. IEEE, 2021. 1

[43] Yichun Shi, Peng Wang, Jianglong Ye, Long Mai, Kejie Li, and Xiao Yang. Mvdream: Multi-view diffusion for 3d generation. In *The Twelfth International Conference on Learning Representations*, 2024. 3

[44] Chaoyue Song, Jiacheng Wei, Chuan Sheng Foo, Guosheng Lin, and Fayao Liu. Reacto: Reconstructing articulated objects from a single video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5384–5395, 2024. 2

[45] Chaoyue Song, Jianfeng Zhang, Xiu Li, Fan Yang, Yiwen Chen, Zhongcong Xu, Jun Hao Liew, Xiaoyang Guo, Fayao Liu, Jiashi Feng, and Guosheng Lin. Magicarticulate: Make your 3d models articulation-ready. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. 2

[46] Jiayi Su, Youhe Feng, Zheng Li, Jinhua Song, Yangfan He, Botao Ren, and Botian Xu. Artformer: Controllable gen-

eration of diverse 3d articulated objects. *arXiv preprint arXiv:2412.07237*, 2024. 3, 5

[47] Fangyin Wei, Rohan Chabra, Lingni Ma, Christoph Lassner, Michael Zollhöfer, Szymon Rusinkiewicz, Chris Sweeney, Richard Newcombe, and Mira Slavcheva. Self-supervised neural articulated shape and appearance models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15816–15826, 2022. 2

[48] Di Wu, Liu Liu, Zhou Linli, Anran Huang, Liangtu Song, Qiaojun Yu, Qi Wu, and Cewu Lu. Reartgs: Reconstructing and generating articulated objects via 3d gaussian splatting with geometric and motion constraints. *arXiv preprint arXiv:2503.06677*, 2025. 2

[49] 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. 1

[50] Runsen Xu, Xiaolong Wang, Tai Wang, Yilun Chen, Jiangmiao Pang, and Dahua Lin. Pointllm: Empowering large language models to understand point clouds. In *European Conference on Computer Vision*, pages 131–147. Springer, 2024. 4, 5

[51] Junkai Yan, Yipeng Gao, Qize Yang, Xihan Wei, Xuansong Xie, Ancong Wu, and Wei-Shi Zheng. Dreamview: Injecting view-specific text guidance into text-to-3d generation. In *European Conference on Computer Vision*, pages 358–374. Springer, 2024. 3

[52] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Huiwen Chang, Deva Ramanan, William T Freeman, and Ce Liu. Lasr: Learning articulated shape reconstruction from a monocular video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15980–15989, 2021. 2

[53] Yuxin Yao, Zhi Deng, and Junhui Hou. Riggs: Rigging of 3d gaussians for modeling articulated objects in videos. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025. 2

[54] Qiaojun Yu, Xibin Yuan, Junting Chen, Dongzhe Zheng, Ce Hao, Yang You, Yixing Chen, Yao Mu, Liu Liu, Cewu Lu, et al. Artgs: 3d gaussian splatting for interactive visual-physical modeling and manipulation of articulated objects. *arXiv preprint arXiv:2507.02600*, 2025. 2

[55] Can Zhang and Gim Hee Lee. Iaao: Interactive affordance learning for articulated objects in 3d environments. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2025. 2