ARTICRAFT: An Agentic System for Scalable Articulated 3D Asset Generation

Matt Zhou[1#*] **Ruining Li**[2*] **Xiaoyang Lyu**[1*] **Zhaomou Song**[1*] **Zhening Huang**[1*]
Chuanxia Zheng[3] **Christian Rupprecht**[2] **Andrea Vedaldi**[2] **Shangzhe Wu**[1]
[1]University of Cambridge [2]University of Oxford [3]Nanyang Technological University
articraft3d.github.io



Figure 1: We introduce ARTICRAFT, an agentic system for scalable articulated 3D asset generation. Using ARTICRAFT, we created Articraft-10K, a large-scale dataset of over 10K articulated 3D assets spanning a wide range of everyday object categories.

Abstract

A bottleneck in learning to understand articulated 3D objects is the lack of large and diverse datasets. In this paper, we propose to leverage large language models (LLMs) to close this gap and generate articulated assets at scale. We reduce the problem of generating an articulated 3D asset to that of writing a program that builds it. We then introduce a new agentic system, ARTICRAFT, that writes such programs automatically. We design a programmatic interface and harness to help the LLM do so effectively. The LLM writes code against a domain-specific SDK for defining parts, composing geometry, specifying joints, and writing tests to validate the resulting assets. The harness exposes a restricted workspace and interface to the LLM, validates the resulting assets, and returns structured feedback. In this way, the LLM is not distracted by details such as authoring a URDF file or managing a complex software environment. We show that this produces higher-quality assets than both state-of-the-art articulated-asset generators and general-purpose coding agents. Using ARTICRAFT, we build Articraft-10K, a curated dataset of over 10K articulated assets spanning 245 categories, and show its utility both for training models of articulated assets and in downstream applications such as robotics simulation and virtual reality.

Project lead. $*$ Core technical contribution.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Articulated objects, such as cabinets, pliers, desk lamps, strollers, folding chairs, and industrial tools, are ubiquitous in the real world and thus important to applications in 3D content creation, gaming, physical design, and robotics. Because their function depends on the motion of their parts as much as on their geometry, modelling shape alone is insufficient: a useful representation must also capture part hierarchy, joints, and range of motion.

A central obstacle in this area is data. Existing datasets of articulated assets are small, narrow in coverage, and uneven in quality [30, 53, 49, 31, 11, 48]. In particular, many object categories of practical importance are underrepresented. As a result, learning-based methods overfit the available data and fail to generalise to new categories [22].

One way to close this gap is to *generate* articulated assets synthetically. This is itself difficult, but recent agentic systems suggest a path forward. Tools such as Claude Code [1] and Codex [36] can plan, write, debug, and iteratively refine software with little human supervision. We hypothesise that these advances transfer to 3D generation, because articulated objects share the recursive, compositional structure of programs. Designing an object is a sequence of decisions: decompose the object into parts, decide how the parts connect, specify joints and motion limits, instantiate geometry, validate behaviour, and revise on failure. This workflow resembles agentic programming far more than one-shot generation.

To test this hypothesis, we introduce ARTICRAFT, an agent that builds articulated 3D objects at scale by generating code. We design ARTICRAFT around three principles. It should be *automatic*: given an object description, it produces an articulated asset without manual intervention. It should be *lightweight*: to keep per-asset inference cost low and make large-scale data generation feasible, it avoids heavy external graphics software (e.g., Blender) and image-based feedback. It should be *expressive*: it covers a wide range of articulated categories, including complex mechanisms.

Our key technical innovation is the design of the agent, which rests on two components: an *agent harness* and an LLM-friendly *Software Development Kit* (SDK). Together, they enable off-the-shelf large language models (LLMs) to generate articulated 3D assets without retraining, drawing on the LLM's coding ability and its prior knowledge of how everyday objects are structured and move. This design distinguishes our approach from prior LLM-based methods for articulated asset generation [55, 13, 20].

The SDK makes it easier for the LLM to understand how to generate assets. The LLM is asked to write a single program against this SDK; once executed, the program produces a complete articulated asset. The SDK is focused, expressive, and LLM-friendly: it spans a wide range of articulated categories while staying close enough to familiar coding patterns that generation remains reliable. It exposes both low-level primitives (e.g., adding a cylinder) and high-level abstractions (e.g., adding a hinge), keeping programs compact and readable, and it lets the LLM write and execute object-specific validation code to test structural integrity.

The harness turns the LLM into an iterative agent rather than a single-pass generator. The harness exposes a minimal workspace and interface: edit a single program, execute it, and receive or request feedback on the resulting 3D asset. Its design encodes deliberate choices about how much context to expose, how that context is structured, how generated assets are validated, and when to stop iterating.

Together, the focused SDK and minimal harness keep per-asset inference cost low, which is what makes large-scale generation practical. This design produces realistic articulated objects across a diverse set of categories. It outperforms both prior work in this area and general-purpose coding agents such as Codex [37] and Claude Code [1]. It is also substantially more lightweight, avoiding the heavy external graphics software (e.g., Blender) that prior approaches often rely on. Using ARTICRAFT, we build *Articraft-10K*, a curated dataset of over 10K articulated objects spanning 245 categories. As one application, we retrain *Particulate* [22], a model that estimates the articulated structure of 3D objects, and obtain a substantial performance boost. We also showcase applications in robotics simulation and virtual reality.

The Articraft-10K dataset is publicly available on 🤗 Hugging Face.

Table 1: Comparison between Articraft-10K and existing articulated or physics-annotated 3D object datasets. For datasets that also contain non-articulated assets, the counts refer to the portion of articulated or interactive objects, where reported.

Dataset	# Categories	# Assets	Source of 3D Geometry
PartNet-Mobility [53]	46	2.3K	PartNet [32]
AKB-48 [31]	48	2.0K	Real scanning
GAPartNet [11]	27	1.2K	PartNet-Mobility [53], AKB-48 [31]
GRScenes [48]	22	1.8K	Human artist design
Infinigen-Sim [18]	18	20K	Procedural generation
PhysXNet [3]	24	26K	PartNet [32]
PhysXNet-XL [3]	11	6M	Procedural generation
PhysX-Mobility [4]	47	2K	PartNet-Mobility [53]
RoboCasa365 [34]	12	0.5K	Human artist design
Articraft-10K (Ours)	245	>10K	Agentic generation

2 Related Work

Generating articulated objects. Recent work has begun to generate articulated 3D assets directly. NAP [21] and CAGE [28] propose ad-hoc representations of articulated objects that can be generated using denoising diffusion [44]. ArtFormer [45] and MeshArt [9] do so with a token-based representation and an auto-regressive generative transformer. URDFormer [6], URDF-Anything [24] and URDF-Anything+ [52] learn to map images to URDF proxy representations, the latter building on AutoPartGen [5] to extract a URDF object sequentially, one part at a time. Real2Code [58] starts by reconstructing an image in 3D and performing part segmentation and then uses LLM-based code generation to infer the articulation parameters. SINGAPO [29] generates articulated object parts from a single image. PhysX-3D [3] augments the structured 3D latents [54] to generate physical attributes in addition to 3D shape and PhysX-Anything [4] predicts simulation-ready geometry, articulation, and physical attributes from a single image. More similar to our approach is Articulate-Anything [20], which uses VLM agents to generate Python code to construct articulated URDF assets from text, images, or videos. However, they rely on retrieving part meshes from an existing 3D asset library, which constrains category diversity. A more recent work, ArtiCAD [43], generates articulated CAD assemblies with a multi-agent pipeline, but relies on visual feedback from multi-view renders and joint-motion keyframes during review. In contrast, ARTICRAFT is entirely code-based, avoiding image-based feedback and keeping large-scale generation lightweight.

Programmatic and agentic 3D design. Several prior works generate 3D objects by controlling CAD software. ShapeAssembly [16] introduces an SDK to simplify interfacing with CAD software and target it with an ad-hoc code generator. ShapeMOD [17] further discovers reusable programmatic primitives in this space. DeepCAD [51] and Text2CAD [19] generate suitably-encoded CAD commands using an autoregressive transformer. Text-to-CadQuery [55] and CAD-Coder [13] write Python code against the CadQuery API. Our agent targets a superset of these APIs and produces articulated, simulation-ready assets instead of static ones.

Our method is also related to language agents that interleave reasoning, action, tool use, and feedback, especially ReAct [57], Reflexion [42], SWE-agent [56], and code-as-action methods such as PAL [10] and Code as Policies [25]. SWE-agent, in particular, shows that LLMs benefit from using software interfaces that are specially designed for the task and that provide feedback in a well-structured format. We follow the same principle, including in the form of a domain-specific SDK that the agent can use to write a program.

Articulated assets and simulation datasets. PartNet-Mobility [53] is perhaps the most widely-used dataset of articulated objects to date based on synthetic assets. AKB-48 [31] starts instead from scans of real objects, and GAPartNet [11] extends PartNet with annotations for part-based affordances. GRScenes, introduced as part of GRUtopia [48], generates entire scenes instead of single objects. RoboCasa365 [34] provides simulated environments for robotics with more than a hundred object categories, of which only a dozen are articulated. Infinigen-Sim [18] follows Infinigen [41] and manually defines procedural generators to generate articulated 3D assets instead of scenes. PhysX-3D [3] introduces PhysXNet, a collection of articulated 3D objects created semi-automatically. We use our new agent to create Articraft-10K, a large collection of curated articulated assets with substantially broader category coverage than these prior works (Table 1).

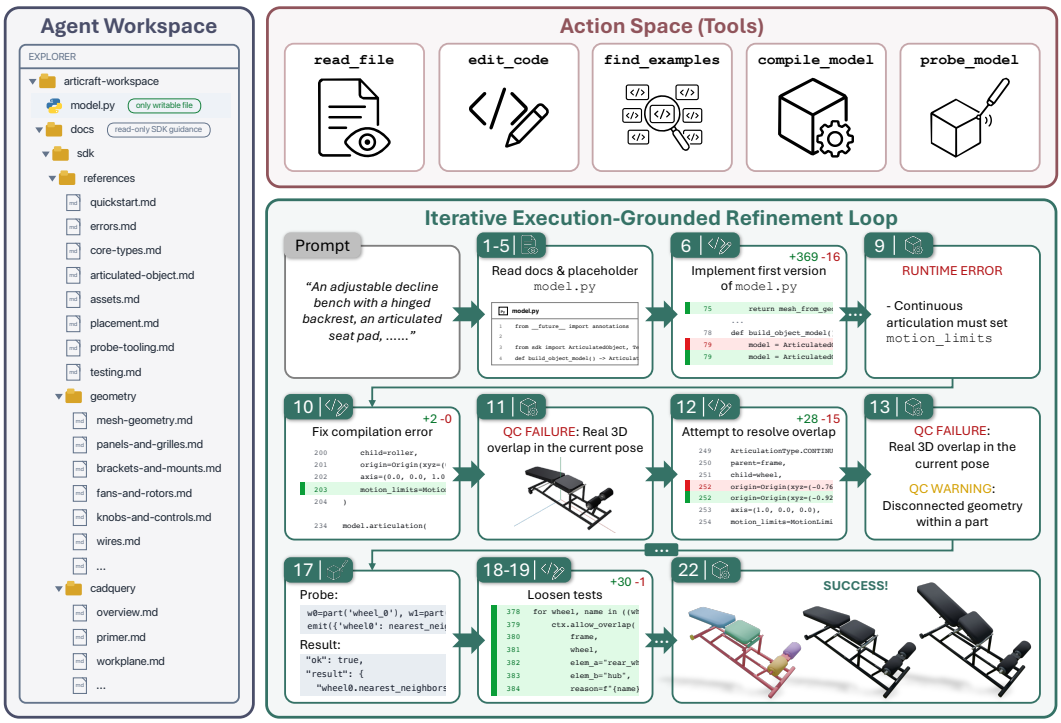


Figure 2: Overview of ARTICRAFT. The agent operates in a restricted workspace with one writable `model.py`, read-only SDK documentation, curated examples, and a small action space. At each iteration, the LLM edits the program, compiles or probes the current asset through the harness, and uses the returned validation signals and feedback for the next revision. See Section 3 for details.

3 ARTICRAFT

We introduce ARTICRAFT, an agent that writes programs to build articulated 3D assets. Given a natural-language description x and, optionally, a reference image of the object, ARTICRAFT writes a Python program y which, once executed, outputs an articulated 3D asset a. The asset consists of a URDF[1] containing 3D meshes, semantic parts, and articulated joints with their axes and motion ranges.

ARTICRAFT builds 3D assets iteratively, following the reasoning-action-observation pattern of language agents [57, 42, 47]. The agent is an off-the-shelf LLM E with coding capabilities. The LLM is given a system prompt p (see Section C) that specifies the object generation task and the programmatic interface and harness that must be used to solve the task. A harness exposes to the LLM a workspace and interface to manipulate the program. The workspace maintains a state consisting of the current program y_t, the current asset a_t, as well as a history h_t of past revisions. The LLM $E(p, x, h_t)$ takes the system prompt p, the user prompt x, and the current history h_t and outputs one or more commands for the harness. These commands act on the current state (y_t, a_t) by editing the program, compiling it into a new asset, or probing the asset to obtain feedback. Thus the harness C outputs a new state (y_{t+1}, a_{t+1}) and feedback s_{t+1}, which are appended to the history for the next iteration:

$$(y_{t+1}, a_{t+1}, s_{t+1}) = C(E(p, x, h_t), y_t, a_t), \quad h_{t+1} = h_t \cup \{(y_{t+1}, a_{t+1}, s_{t+1})\}.$$

The process terminates when the LLM does not issue further editing commands and when the validation criteria are met. The last version of the asset a_T is accepted as final output.

A key innovation of ARTICRAFT is to provide the LLM backend with a programmatic interface (Section 3.1) and a harness (Section 3.2) specific to articulated 3D design (Fig. 2). This follows the principle that agent-computer interfaces should be task-specific instead of generic [56] and significantly improves the effectiveness of the backend, which is otherwise a general purpose off-the-shelf coding model. In this manner, ARTICRAFT does not need to use *visual* feedback used in

[1] Unified Robot Description Format, a commonly used XML format for representing articulated 3D assets.

```python
def build_object_model() -> ArticulatedObject:
  model = ArticulatedObject(name="desk_lamp")
  # Define materials
  base_finish = model.material("base_finish",
    rgba=(0.16, 0.17, 0.19, 1.0))
  ...
  # Add a part called "base"
  base = model.part("base")
  base.visual(
    Cylinder(radius=base_radius, length=base_thickness),
    origin=Origin(xyz=(0.0, 0.0, base_thickness / 2.0)),
    material=base_finish,
    name="base_plate",
  )
  ...
  base.inertial = Inertial.from_geometry(
    Cylinder(radius=base_radius, length=base_thickness),
    mass=2.4,
    origin=Origin(xyz=(0.0, 0.0, base_thickness / 2.0)),
  )
  # Add other parts
  lower_arm = model.part("lower_arm")
  ...
  # Add joints between parts
  model.articulation(
    "base_to_lower_arm",
    ArticulationType.REVOLUTE,
    parent=base,
    child=lower_arm,
    origin=Origin(xyz=(0.0, 0.0, shoulder_z)),
    axis=(0.0, -1.0, 0.0),
    motion_limits=MotionLimits(lower=-0.35, upper=1.15,
      effort=18.0, velocity=1.6),
  )
  ...
  return model
```



Figure 3: Excerpt from a `model.py` program authored by ARTICRAFT for an articulated desk lamp.

prior work [20], which is expensive to produce and use. Instead, the harness and programmatic interface provide specialised tools for authoring and checking the geometry of the object directly and efficiently.

3.1 The ARTICRAFT Programmatic Interface

Each 3D asset a generated by ARTICRAFT is defined by a program y consisting of a single Python file, `model.py` (see the example in Fig. 3). The file exposes two entry points: `build_object_model()` to construct the articulated object, and `run_tests()` to record prompt-specific geometric checks and explicit decisions to relax some of these. It then binds the constructed object to the Python variable `object_model`, which the harness can use to extract the generated asset. An implicit *contract* separates object authoring from details of system execution, such as file management, URDF export, and validation. The agent only needs to write code against the SDK given below to specify the object parts, their geometry and articulation, and what object-specific tests should hold, and the harness handles the rest. This keeps the editable target expressive but small.

Defining Geometry and Parts. The program `model.py` uses the SDK to build the object. At the top level, the program constructs an `ArticulatedObject` and populates it with named `Parts`, which provide the semantic scaffold for the asset. Parts can be connected by joints and named as targets for tests and geometric probes. The SDK also contains a variety of tools for defining the geometry of the parts, from low-level generation of primitives like boxes, cylinders, and spheres, to invoking CAD-like tools (exposing CadQuery [7]) and high-level procedural generators that output complex structures such as supports, panels, hinges, wheels, grilles, and swept profiles. These tools are composable and category-agnostic to create a wide variety of object types and shapes. By providing both low- and high-level primitives, the agent can write more concise programs that are more token efficient and more likely to be correct while retaining fine-grained control over the geometry. The SDK also includes a tool to find examples of code snippets that match a natural language description, which the agent can use to retrieve relevant code patterns from a curated example library.

Defining Articulations. Each `ArticulatedObject` stores articulated joints between its named `Parts`, with SDK support for revolute, prismatic, continuous, and fixed joints with explicit origins, axes, and motion limits. These joints are represented as `Articulation` objects, which record the parent and child parts, joint type, origin, axis, and motion limits. This ties each articulated joint to the parts and geometry, allowing both to be revised together as the agent iterates on the design. For example, a drawer needs both rails in the right place and a prismatic joint aligned with those rails. The compiled URDF preserves this kinematic structure, so the output is not only visual geometry but also a structured representation of how parts move. It can therefore be consumed by robotics simulators, interactive viewers, and downstream learning pipelines.

Self Validation. As noted below, the harness automatically performs a number of default checks to validate the generated assets a, returning feedback to the agent. These include detecting runtime errors when the program executes, disconnected parts, and parts that overlap. However, the integrity of the generated asset often depends on subtle object-specific relations: a drawer should remain seated in its rails, a hinged lid should clear its base through its motion, and a knob stem should stay seated in its socket. `model.py` thus includes an entry point `run_tests()` that the agent can use to test for such object-specific properties, and which is executed by the harness after its own checks. The system prompt and curated examples tell the agent how to do this: instantiate `TestContext(object_model)`, resolve named parts or visual elements, call assertion or exemption helpers, and return `ctx.report()`. Assertion helpers express geometric constraints such as contact, gap, overlap, containment, and pose-dependent relationships between parts. For example, `expect_contact(...)` can be used to check that a leaf of a hinge is in touch with the frame of a

Table 2: Restricted action space used by ARTICRAFT. The interface exposes only actions that directly support articulated object authoring and repair.

Family	Tool name(s)	Edits?	Purpose
Read	`read_file`	×	Read exact text from `model.py` and SDK references.
Edit	`apply_patch`, `replace`, `write_file`	✓	Apply local edits or controlled replacements to the bound object program.
Examples	`find_examples`	×	Retrieve curated examples for geometry, mechanism, placement, and test patterns.
Compile/QC	`compile_model`	×	Execute the current program, export URDF when successful, run baseline QC and authored tests, and return structured signals.
Probe	`probe_model`	×	Run a read-only snippet over the current `object_model` and return JSON measurements.

cabinet and `expect_within(...)` that a drawer remains engaged with its rails. Exemption calls such as `allow_overlap(...)` and `allow_isolated_part(...)` instruct the harness to intentionally ignore some of its own automated checks, for instance to tolerate local interpretation or a part which is intentionally freestanding.

3.2 The ARTICRAFT Harness

The harness provides the agent with a restricted workspace and interface to manipulate the program.

Restricted Workspace. Differently from general coding agents, the ARTICRAFT agent does not operate directly on a complex codebase. Instead, the harness presents to the agent a workspace containing only one writable artifact, `model.py`, plus read-only access to the SDK documentation and curated examples, as illustrated in Fig. 2.

Restricted Action Space. Also differently from general coding agents, the ARTICRAFT agent cannot execute arbitrary shell commands, navigate complex code repositories, or refactor multiple files. Instead, the harness removes these irrelevant degrees of freedom and only offers a small number of commands to read the SDK documentation, patch or replace text in the program, retrieve code examples, compile the program, and run read-only geometric probes, as summarized in Table 2. Each non-editing tool, furthermore, returns feedback that the harness passes back to the agent. The simplicity of the action space makes the ARTICRAFT harness friendly for non-frontier LLMs, allowing for cheap and scalable synthetic data generation.

Execution Feedback. As the harness executes the commands from the agent, it provides feedback on execution, including asset quality. This feedback is structured as `failure`, `warning`, and `note` signals instead of being provided as raw logs. Failure signals include program execution errors and failed validation tests, including those authored by the agents and the default ones defined by the harness; warnings report non-fatal geometric, structural, or code quality issues; notes record context such as exemptions issued by the LLM. When the agent finds the feedback to be insufficient, it can call `probe_model`, a read-only tool that looks at the current `object_model` and returns a variety of measurements. These can be used for adjusting properties such as distances, overlaps, containment relations, or pose of the parts.

Trace and Metadata Logging. When the harness runs, it stores an auditable record of the run: the input prompt, conversation messages, tool calls, compile and probe feedback, final `model.py`, provider and model identifiers, turn and cost statistics, output artifact paths, and hashes of the prompt and generated program. Dataset records can also store curator ratings, which are used for dataset filtering and analysis rather than for generation-time feedback.

3.3 Image-conditioned Generation

When a reference image is provided, it is treated as the primary source of truth for geometry, proportions, articulation, and materials, overriding generic object priors. The image persists throughout the edit–execute–repair loop, enabling each iteration to re-ground edits in the same visual evidence and prevent drift toward category-level defaults. Once the agent produces a URDF with approximately colored materials, we can further refine the physically based rendering (PBR) materials of the asset using the approach introduced in LiteReality [14]. The final output is a fully articulated object with full PBR materials that closely resembles the one in the reference image, as illustrated in Fig. 8.

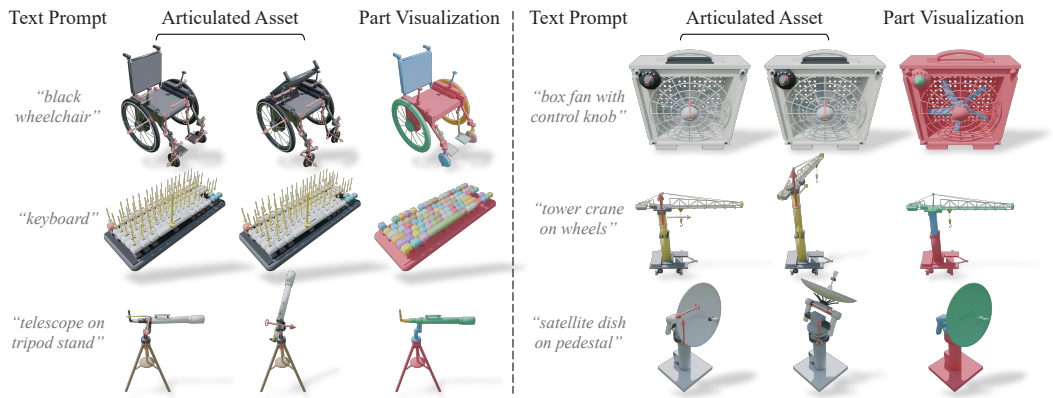


Figure 4: Examples of articulated 3D assets in the Articraft-10K dataset generated by ARTICRAFT.



Figure 5: Comparison of articulated asset generation with different methods.

4 Articraft-10K

We use ARTICRAFT to create Articraft-10K, a large, curated dataset of over 10K articulated 3D models (Figs. 1 and 4). Articraft-10K contains 245 object categories, which we further map into 15 super-categories and visualize in Fig. 11a, together with a word cloud. Each asset ships with a URDF file, the corresponding `model.py`, as well as the full trace of the agent (showing reasoning, feedback, and tool use; see Fig. 2 and the appendix for examples). In the future, these traces can be used to post-train open-source language models through supervised fine-tuning, which should help to reduce the gap between open and closed models when it comes to 3D generation.

Dataset Construction. To construct Articraft-10K, we began by selecting a large set of object categories that the model can generate well. To do so, we first started to explore the capabilities of our agent by prompting it manually. By looking at successes and failure cases, we created a set of guidelines to identify further categories that were likely to work well, fed the guidelines to an LLM to propose new categories, and manually reviewed and filtered those. Using this process, we identified 245 suitable categories such as waffle makers, drones, stationary exercise bikes, stand mixers, tripod-mounted devices, and sewing boxes with hinged lids. Then, we developed further guidelines for the LLM to generate prompts for each category, again with an eye to the agent's capabilities and limitations. These prompts were then given to the ARTICRAFT agent to produce Articraft-10K. Typical failure cases are included in Section B.5 and Fig. 18.

We further filtered the generated assets via a manual *rating process*. Each generated object received a score from 1 to 5 according to three criteria: (1) realism of the overall geometry and its individual components, (2) presence of articulated motions where they are expected, and (3) whether the articulations adhere to basic physical constraints (e.g., no floating links or implausible movements). For each minor violation of these criteria, one point was deducted from the initial score of 5. Objects with extreme violations of any criterion, or with a final score below 4, were excluded from the dataset. Across the GPT-5.4, GPT-5.5, and Gemini 3.1 Pro generation runs, 10,018 out of 10,909 generated assets passed this filter, for an overall retention rate of 91.8%. See Section A.2 for further details.

Table 3: Articraft-10K assets can boost the performance of feed-forward 3D articulation estimation models. We train Particulate [22] with augmented training data from Articraft-10K and evaluate the resulting Particulate-ARTICRAFT model on the Lightwheel benchmark [26] following the same protocol.

	Rest-Pose Segmentation			Articulated Geometry		
Model	gIoU↑	PC↓	mIoU↑	gIoU↑	PC↓	OC↓
Particulate [22]	0.332	0.168	0.576	0.305	0.208	0.009
Particulate-ARTICRAFT	**0.394**	**0.144**	**0.607**	**0.361**	**0.179**	**0.008**



Text Prompt: *"a compact folding quadcopter drone"*

Figure 7: Comparison of 3D articulated asset generation with different base models, demonstrating how structural complexity and part completeness vary with model capability and effort level.

5 Evaluating the ARTICRAFT Agent

We begin by assessing the ARTICRAFT against other methods for generating articulated 3D objects and consider four representative baselines: Articulate-Anything [20], PhysX-Anything [4], URDF-Anything+ [52], and Codex [37]. To further investigate the impact of the underlying foundation model, we evaluate two variants of our method powered by different Large Language Models: ARTICRAFT (w/ GPT-5.4) and ARTICRAFT (w/ GPT-5.5).

Experiment Details. As for any generation task, it is difficult to define good automated metrics, so we primarily rely on a user study to evaluate the generation quality. To this end, we construct a benchmark prompt set from the 46 categories in PartNet-Mobility [53], writing 5 prompts for each category. We then generate a reference image using [12] from each benchmark prompt to be input to the image-conditioned methods (i.e., PhysX-Anything and URDF-Anything+). For Codex, we prompt it to generate the articulated assets described by the text prompts in the URDF format, making available to it all programming and web search tools.

To assess the quality of the generated assets, participants were presented with the text prompts alongside the generated 3D assets from all six competing methods (including two variants of ours) in a randomized order. The perceptual study was distributed across 125 non-expert college students with diverse academic backgrounds, with each participant evaluating around 40 randomly assigned objects, yielding a total of 5000 submitted comparisons. Each participant was asked to select and rank the three best results out of six based on prompt alignment and overall quality.

Results. The user study results are summarized in Fig. 6. A key comparison is GPT-5.5 vs ARTICRAFT utilising the same underlying LLM. The difference is striking, as GPT-5.5 alone ranks second to last. This shows the significant effect of providing the LLM with a domain-specific programming interface and harness. Visual comparisons are shown in Fig. 5 with more examples in the supplementary website.

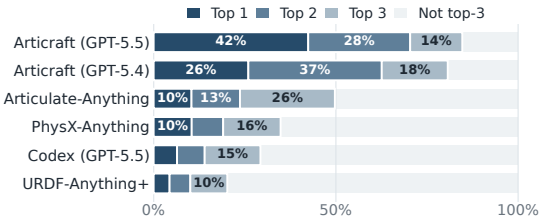


Figure 6: User study results.

Ablation on LLMs. ARTICRAFT can use any coding-capable LLM. In Fig. 7, we compare OpenAI GPT-5.5 [39], Google Gemini 3.1 Pro [12], and Anthropic Claude Opus 4.7 [2] on the same articulated drone prompt, with all models set to high reasoning effort. All recover the requested kinematic structure, but GPT-5.5 produces more visual detail, while Gemini and Claude produce simpler geometry. We also vary GPT-5.5 reasoning effort. Low, medium, and high effort produce the same intended structure with 39, 51, and 78 visual elements, respectively, suggesting that reasoning effort mainly increases geometric and surface detail in this illustrative example. See Section B.4 for the exact prompt and run statistics.

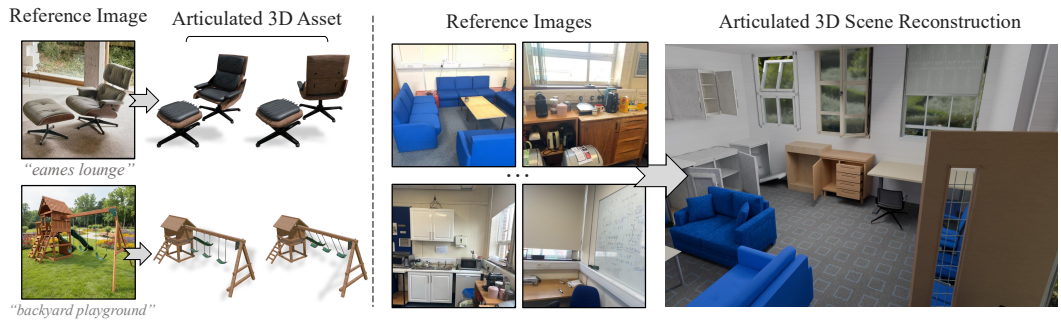


Figure 8: Image-conditioned generation results from ARTICRAFT. Given a reference image, ARTICRAFT produces a fully articulated 3D asset (left). Integrated into LiteReality [14], it also enables articulated room-level reconstructions from indoor scans (right). More results in the supplementary website.

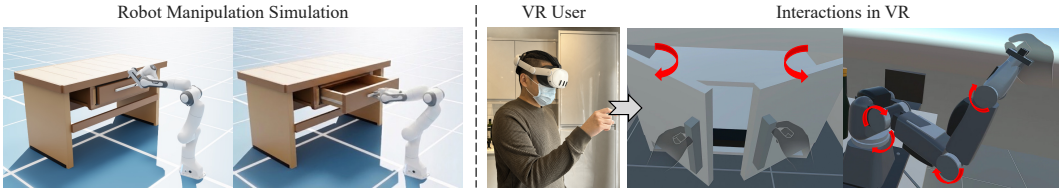


Figure 9: The articulated 3D assets generated by ARTICRAFT can be directly imported into the physics simulator and manipulated with robot arms and VR headsets.

Reconstructing Real Scenes. We build on our agent's ability to be prompted by an image to perform full-room reconstruction following the LiteReality [14] pipeline. We start from an indoor capture from iPhone RoomPlan, which provides each object's 3D position, orientation, and scale. Cropped reference images for each piece of furniture are automatically extracted and used to prompt ARTICRAFT to generate corresponding articulated objects. We then apply the material painting stage (Section 3.3) to recover PBR materials and assemble the generated assets into the parsed scene layout for room reconstruction. The resulting scenes (Fig. 8) are faithful to the original capture while adding articulation, making them ready for interaction and simulation.

6 Evaluating the ARTICRAFT-10K Dataset

Having validated our agent, we now assess the effectiveness of the new Articraft-10K dataset as training data and in applications.

Training Feed-Forward 3D Object Articulation Models. We consider Particulate [22], a recent transformer-based model that predicts 3D parts, kinematic structure, and joint parameters given a single 3D mesh as input. As shown in Table 3, augmenting their training data, namely PartNet-Mobility [53] and GRScenes [48], with Articraft-10K boosts their performance. This shows that the data generated by our agent is very effective to train this type of models. Table 6 provides a per-category breakdown, indicating that the gains from Articraft-10K are particularly pronounced for categories outside the original training distribution.

Using Articraft-10K in Robotic Simulation. We deploy the assets in Articraft-10K in the NVIDIA Isaac Sim [35] environment. For each object, we define a target trajectory for a selected part. We then use standard Inverse Kinematics to control a robotic end-effector to interact with the generated object, for example, using the Franka arm to pull a drawer open (see Fig. 9 for an example). A successful interaction shows that the object is not just visually plausible, but has a valid articulated structure suitable for simulation. More results are provided in the supplementary website.

Virtual Reality. We also deploy the assets in Articraft-10K in a virtual reality (VR) environment (Fig. 9). We simulate interactions between the user's hands and the objects. After detecting a collision, custom scripts control the motion of the joints defined in the URDF model. As shown in the supplementary website, the resulting interactions are natural and realistic.

7 Conclusions

We introduced ARTICRAFT, an agentic system for scalable articulated 3D asset generation. By pairing a task-specific SDK with a restricted execution harness, ARTICRAFT turns articulated asset creation into an edit–execute–repair loop grounded in validation and geometric feedback, without relying on rendered visual inspection. Using this agentic system, we created Articraft-10K, a large-scale dataset of over 10K high-quality articulated 3D assets with source programs, semantic part structures, joint specifications, and generation traces. Our results suggest that the domain-specific programmatic interface and execution harness are key ingredients for scalable and robust articulated asset generation, enabling downstream applications in feed-forward model training, simulation, and VR interaction.

Acknowledgements

Ruining Li is supported by a Toshiba Research Studentship. Chuanxia Zheng is supported by NTU SUG-NAP and National Research Foundation, Singapore, under its NRF Fellowship Award NRF-NRFF17-2025-0009. Christian Rupprecht is supported by an Amazon Research Award and ERC starting grant 'Volute' (No. 101222037). This work is partially supported by the UKRI AIRR programme (ID: u6en), NVIDIA Academic Grant Program using NVIDIA RTX PRO 6000 GPUs, Google Cloud Research Credits program with the award GCP19980904, and ERC CoG 101001212-UNION.

References

[1] Anthropic. Claude Code. `https://docs.anthropic.com/en/docs/claude-code/overview`, 2026. [Online; Accessed 06-May-2026].

[2] Anthropic. Introducing Claude Opus 4.7. `https://www.anthropic.com/news/claude-opus-4-7`, 2026. [Online; Accessed 06-May-2026].

[3] Ziang Cao, Zhaoxi Chen, Liang Pan, and Ziwei Liu. PhysX-3D: physical-grounded 3D asset generation. In *NeurIPS*, 2025.

[4] Ziang Cao, Fangzhou Hong, Zhaoxi Chen, Liang Pan, and Ziwei Liu. PhysX-Anything: simulation-ready physical 3D assets from single image. In *CVPR*, 2026.

[5] Minghao Chen, Jianyuan Wang, Roman Shapovalov, Tom Monnier, Hyunyoung Jung, Dilin Wang, Rakesh Ranjan, Iro Laina, and Andrea Vedaldi. AutoPartGen: Autoregressive 3D part generation and discovery. In *NeurIPS*, 2025.

[6] Zoey Chen, Aaron Walsman, Marius Memmel, Kaichun Mo, Alex Fang, Karthikeya Vemuri, Alan Wu, Dieter Fox, and Abhishek Gupta. URDFormer: A pipeline for constructing articulated simulation environments from real-world images. In *RSS*, 2024.

[7] CadQuery contributors. Cadquery, 2026.

[8] Ye Fang, Zeyi Sun, Tong Wu, Jiaqi Wang, Ziwei Liu, Gordon Wetzstein, and Dahua Lin. Make-it-Real: Unleashing large multimodal model for painting 3D objects with realistic materials. In *NeurIPS*, 2024.

[9] Daoyi Gao, Yawar Siddiqui, Lei Li, and Angela Dai. MeshArt: Generating articulated meshes with structure-guided transformers. In *CVPR*, 2025.

[10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *ICML*, 2023.

[11] Haoran Geng, Helin Xu, Chengyang Zhao, Chao Xu, Li Yi, Siyuan Huang, and He Wang. GAPartNet: Cross-category domain-generalizable object perception and manipulation via generalizable and actionable parts. In *CVPR*, 2023.

[12] Google DeepMind. Gemini 3.1 Pro Model Card. `https://deepmind.google/models/model-cards/gemini-3-1-pro/`, 2026. [Online; Accessed 06-May-2026].

[13] Yandong Guan, Xilin Wang, XiMing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-coder: Text-to-CAD generation with chain-of-thought and geometric reward. In *NeurIPS*, 2025.

[14] Zhening Huang, Xiaoyang Wu, Fangcheng Zhong, Hengshuang Zhao, Matthias Nießner, and Joan Lasenby. LiteReality: Graphics-ready 3D scene reconstruction from RGB-D scans. In *NeurIPS*, 2025.

[15] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. Ditto: Building digital twins of articulated objects from interaction. In *CVPR*, 2022.

[16] R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. ShapeAssembly: Learning to generate programs for 3D shape structure synthesis. In *SIGGRAPH*, 2020.

[17] R. Kenny Jones, David Charatan, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. ShapeMOD: Macro operation discovery for 3D shape programs. In *SIGGRAPH*, 2021.

[18] Abhishek Joshi, Beining Han, Jack Nugent, Yiming Zuo, Jonathan Liu, Hongyu Wen, Stamatis Alexandropoulos, Tao Sun, Alexander Raistrick, Gaowen Liu, Yi Shao, and Jia Deng. Infinigen-Sim: Procedural generation of articulated simulation assets. *arXiv preprint arXiv:2505.10755*, 2025.

[19] Mohammad Sadil Khan, Sankalp Sinha, Sheikh Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *NeurIPS*, 2024.

[20] 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. In *ICLR*, 2025.

[21] Jiahui Lei, Congyue Deng, Bokui Shen, Leonidas J. Guibas, and Kostas Daniilidis. NAP: Neural 3D articulation prior. In *NeurIPS*, 2023.

[22] Ruining Li, Yuxin Yao, Chuanxia Zheng, Christian Rupprecht, Joan Lasenby, Shangzhe Wu, and Andrea Vedaldi. Particulate: Feed-forward 3D object articulation. In *CVPR*, 2026.

[23] Xiaolong Li, He Wang, Li Yi, Leonidas J. Guibas, A. Lynn Abbott, and Shuran Song. Category-level articulated object pose estimation. In *CVPR*, 2020.

[24] Zhe Li, Xiang Bai, Jieyu Zhang, Zhuangzhe Wu, Che Xu, Ying Li, Chengkai Hou, and Shanghang Zhang. URDF-Anything: Constructing articulated objects with 3D multimodal language model. In *NeurIPS*, 2025.

[25] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *ICRA*, 2023.

[26] Lightwheel. Simready: Simulation-ready 3d assets. `https://simready.com/`, 2025. Accessed: 2025.

[27] Jiayi Liu, Ali Mahdavi-Amiri, and Manolis Savva. PARIS: Part-level reconstruction and motion analysis for articulated objects. In *ICCV*, 2023.

[28] Jiayi Liu, Hou In Ivan Tam, Ali Mahdavi-Amiri, and Manolis Savva. CAGE: Controllable Articulation GEneration. In *CVPR*, 2024.

[29] Jiayi Liu, Denys Iliash, Angel X. Chang, Manolis Savva, and Ali Mahdavi-Amiri. SINGAPO: Single image controlled generation of articulated parts in objects. In *ICLR*, 2025.

[30] Jiayi Liu, Manolis Savva, and Ali Mahdavi-Amiri. Survey on modeling of human-made articulated objects. *Computer Graphics Forum*, 2025.

[31] 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 *CVPR*, 2022.

[32] 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 *CVPR*, 2019.

[33] 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 *ICCV*, 2021.

[34] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *ICLR*, 2026.

[35] NVIDIA. Isaac sim, 2025.

[36] OpenAI. Codex CLI. `https://developers.openai.com/codex/cli`, 2026. [Online; Accessed 06-May-2026].

[37] OpenAI. Introducing GPT-5.3-Codex. `https://openai.com/index/introducing-gpt-5-3-codex/`, 2026. [Online; Accessed 06-May-2026].

[38] OpenAI. Introducing GPT-5.4. `https://openai.com/index/introducing-gpt-5-4/`, 2026. [Online; Accessed 06-May-2026].

[39] OpenAI. Introducing GPT-5.5. `https://openai.com/index/introducing-gpt-5-5/`, 2026. [Online; Accessed 06-May-2026].

[40] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

[41] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite photorealistic worlds using procedural generation. In *CVPR*, 2023.

[42] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[43] Yuan Shui, Yandong Guan, Zhanwei Zhang, Juncheng Hu, Jing Zhang, Dong Xu, and Qian Yu. Articad: Articulated cad assembly design via multi-agent code generation. *arXiv preprint arXiv:2604.10992*, 2026.

[44] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.

[45] Jiayi Su, Youhe Feng, Zheng Li, Jinhua Song, Yangfan He, Botao Ren, and Botian Xu. ArtFormer: controllable generation of diverse 3d articulated objects. In *CVPR*, 2025.

[46] Giuseppe Vecchio and Valentin Deschaintre. Matsynth: A modern pbr materials dataset. In *CVPR*, 2024.

[47] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2025.

[48] Hanqing Wang, Jiahe Chen, Wensi Huang, Qingwei Ben, Tai Wang, Boyu Mi, Tao Huang, Siheng Zhao, Yilun Chen, Sizhe Yang, Peizhou Cao, Wenye Yu, Zichao Ye, Jialun Li, Junfeng Long, Zirui Wang, Huiling Wang, Ying Zhao, Zhongying Tu, Yu Qiao, Dahua Lin, and Jiangmiao Pang. GRUtopia: Dream general robots in a city at scale. *arXiv preprint arXiv:2407.10943*, 2024.

[49] Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. Shape2Motion: Joint analysis of motion parts and attributes from 3D shapes. In *CVPR*, 2019.

[50] Yijia Weng, He Wang, Qiang Zhou, Yuzhe Qin, Yueqi Duan, Qingnan Fan, Baoquan Chen, Hao Su, and Leonidas J. Guibas. CAPTRA: Category-level pose tracking for rigid and articulated objects from point clouds. In *ICCV*, 2021.

[51] Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: a deep generative network for computer-aided design models. In *ICCV*, 2021.

[52] Zhuangzhe Wu, Yue Xin, Chengkai Hou, Minghao Chen, Yaoxu Lyu, Jieyu Zhang, and Shanghang Zhang. URDF-Anything+: End-to-end generation for simulation-ready articulated assets. In *NeurIPS*, 2026.

[53] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *CVPR*, 2020.

[54] Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3D latents for scalable and versatile 3d generation. In *CVPR*, 2025.

[55] Haoyang Xie and Feng Ju. Text-to-CadQuery: A new paradigm for CAD generation with scalable large model capabilities. *arXiv preprint arXiv:2505.06507*, 2025.

[56] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, 2024.

[57] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[58] Zhao Mandi, Yijia Weng, Dominik Bauer, and Shuran Song. Real2Code: Reconstruct Articulated Objects via Code Generation. In *ICLR*, 2025.

A Additional Statistics

A.1 Articraft-10K Dataset

Figure 11a shows the distribution of objects across the 15 super-categories in Articraft-10K. Figure 11b plots the average cost of each category against the average number of turns the ARTICRAFT agent required, revealing a positive correlation between the two quantities. Figure 12 presents a word cloud generated from frequencies of all object names, with larger font sizes representing higher frequency.

Figure 13 plots the distributions of generation cost, number of turns, and number of links (parts) across all objects in the dataset. Figure 14 summarizes the mesh statistics of the dataset, including the distributions of vertices, triangles, and edges per object. Finally, Figure 15 shows the distribution of URDF joint counts grouped by joint type for objects in Articraft-10K.

A.2 Retention After Manual Rating

We report curation retention as the fraction of generated assets that received a final manual rating of at least 4. Table 4 breaks down the retention statistics according to the model used for generation.

Table 4: Manual curation retention by backend. Retained assets are those rated 4 or 5.

Backend	Generated	Retained	Retention rate
GPT-5.4	6,601	5,903	89.4%
GPT-5.5	4,010	3,828	95.5%
GPT-5.4/GPT-5.5 total	10,611	9,731	91.7%
Gemini 3.1 Pro	298	287	96.3%

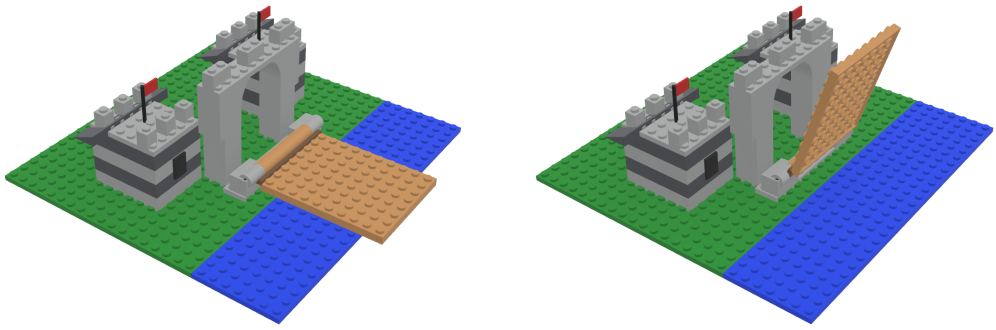
A.3 Compute Requirements for ARTICRAFT Generation

The compute requirements of ARTICRAFT are modest because generation does not train a model and does not require rendered visual feedback. The expensive inference step is provided by external LLM API backends, while the local harness runs `model.py`, CAD and mesh construction, URDF export, authored tests, and quality-control checks on CPU workers. We used heterogeneous CPU workers for these local steps; no GPU is required for ARTICRAFT generation, dataset materialization, or the compile/QC loop. Each object is generated independently, so dataset construction parallelizes by distributing records across CPU workers, with worker memory mainly determined by the temporary CadQuery and mesh-processing state of a single object. Wall-clock time for a batch is therefore dominated by LLM provider latency, the number of agent turns, and the chosen number of parallel workers, rather than by local accelerator compute.

We log the provider, model, number of turns, API cost, generated program, compile outputs, and curator rating for each record. Figures 11b and 13 summarize the per-object generation cost and turn counts for Articraft-10K. The same lightweight CPU-only local pipeline was used for retained objects, filtered objects, and exploratory runs; Table 4 reports both generated and retained counts for the main generation runs used to construct the dataset. Downstream demonstrations such as Isaac Sim and VR use their respective simulator or display hardware, but these are not required to reproduce the core ARTICRAFT generation pipeline.

A.4 API Cost and Token Usage

Table 5 summarizes the API usage recorded by the ARTICRAFT harness for the main LLM backends used in Articraft-10K construction. Costs are computed from the provider pricing active during generation and include cached-prompt discounts when available. The cost analysis covers all retained objects and most filtered generation attempts: 10,880 generated attempts with cost logs out of the 10,909 attempts in Table 4. Across these cost-logged attempts, the total API cost was approximately \$12.39K, or \$1.14 per generated attempt on average. The retained Articraft-10K objects account for approximately \$11.33K, or \$1.13 per retained object on average. Prompt caching was important for scalability: 85.7% of prompt tokens in the main-backend cost logs were served from cache.



(a) Drawbridge lowered. (b) Drawbridge raised.

Figure 10: Example of the ARTICRAFT SDK's expressivity and style controllability. The same block-based gatehouse is rendered in two drawbridge states, showing that the generated asset controls both construction idiom and articulation.

Table 5: API cost and turn statistics for the main generation backends. Costs are in USD and are computed from available per-record cost logs.

Backend	Cost logs	Retained	Total cost	Mean/med. cost	Mean/med. turns
GPT-5.4	6,572	5,903	$6,693.89	$1.019 / $0.887	16.9 / 15
GPT-5.5	4,010	3,828	$5,305.18	$1.323 / $1.240	16.4 / 15
Gemini 3.1 Pro	298	287	$387.91	$1.302 / $0.711	19.5 / 10
Total	10,880	10,018	$12,386.99	$1.139 / $1.030	16.8 / 15

B Additional Results

B.1 Additional Examples of the Generated Articulated Assets in Articraft-10K

Figure 19 provides additional examples of assets generated by ARTICRAFT for Articraft-10K. The examples illustrate the breadth of object categories and scales covered by the dataset, ranging from small hand-held objects such as syringes and bottles, to household and workshop objects such as sewing boxes, desktop PC towers, miter saws, and stove tops, to larger structures such as drawbridges, barriers, windmills, and lighthouse beacons. They also show a range of articulation types, including hinged doors and lids, sliding panels, folding linkages, rotating wheels, gears, beacons, and propeller-like assemblies, telescoping or prismatic motion, and swinging mechanisms. For each prompt, we show rendered asset states together with a colored part visualization, highlighting that the generated objects contain explicit semantic parts and articulated structure rather than only static geometry.

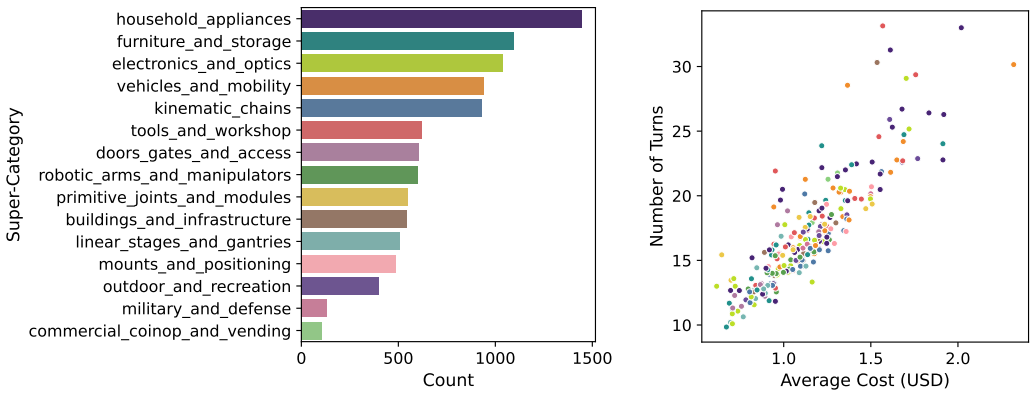
SDK expressivity. The ARTICRAFT SDK is not tied to a single visual style or object family: prompts can specify both mechanism and construction idiom. Figure 10 illustrates this with a block-based gatehouse whose style, parts, and drawbridge motion are all explicitly controlled.

B.2 Additional Details on Image-conditioned Generation

Object-level Reconstruction. Image conditioning in ARTICRAFT has two goals: anchoring geometry to the reference photograph, and recovering PBR materials so the result is visually realistic.

Geometry conditioning. When the user supplies a reference image alongside the prompt, the harness attaches it to the agent's initial user turn, and the system prompt elevates it to the primary ground truth. Because the image persists in the multi-turn context, every compile–probe–edit cycle re-grounds itself in the same visual evidence, keeping local edits consistent with the photograph rather than drifting toward type-priors. The agent outputs a URDF with flat PBR materials, where base colors are approximated from the input image

Material Painting. We follow LiteReality [14]'s retrieval-based strategy with albedo-only optimization, drawing PBR maps from a curated material database filtered from [46] rather than synthesizing



(a) Object count by super-categories.

(b) Average cost and turns per category.

Figure 11: Statistics of Articraft-10K: (a) Number of objects in each super-category. (b) Average cost and number of turns for each object category, color-coded by super-category.

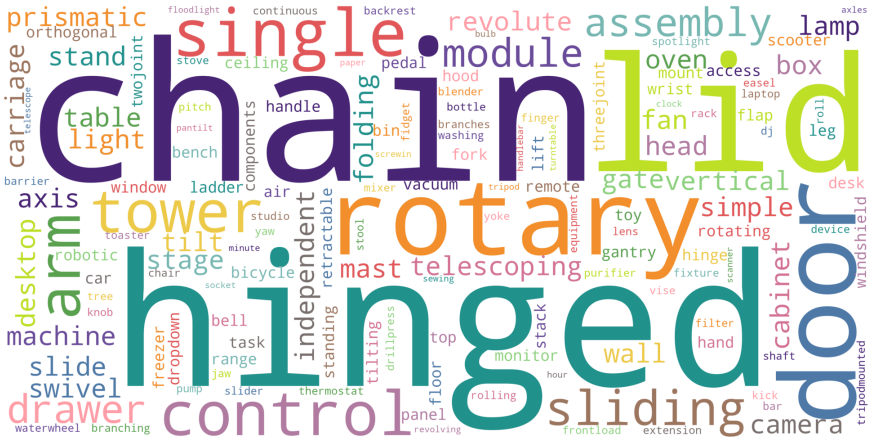


Figure 12: Word cloud of object names in Articraft-10K.

them from scratch. Retrieval is hierarchical: the agent narrows the candidate pool through three layers of material categories using language cues [8], then ranks the shortlist with DINOv2 [40] visual features, followed by a final LLM-based selection step. We then apply LiteReality's albedo-only optimization: the base color's HSV centroid is shifted toward the target while local deviations are preserved, retaining grain and weathering at the chosen color. Because color dominates perceived match quality, we add a color-refinement loop that re-renders the object, compares it against the reference image, and adjusts until the color is correct.

Together, these two stages let ARTICRAFT turn an arbitrary image into an articulated asset with faithful geometry, valid joints, and complete PBR materials, as shown in Fig. 16.

Scene-level Reconstruction. We extend ARTICRAFT to full-room reconstruction by plugging it into the LiteReality pipeline [14], which converts RGB-D scans, captured via Apple RoomPlan with per-object bounding boxes, orientations, and scales, into graphics-ready scenes through three stages: object reconstruction, material painting, and scene integration. LiteReality's original object stage relies on retrieval from a curated CAD database; we replace it with Articraft so that objects are *generated* with articulation rather than retrieved as rigid meshes, and we fold the material-painting stage directly into Articraft's image-conditioning pipeline.

Because ARTICRAFT can struggle on highly irregular or unconventionally designed objects, we add a simple per-object switch: objects flagged as articulated are generated with ARTICRAFT, while the rest fall back to LiteReality's retrieval path. Given an RGB-D capture, the pipeline first crops the most visible reference image for each detected object, then routes each crop accordingly. The articulated

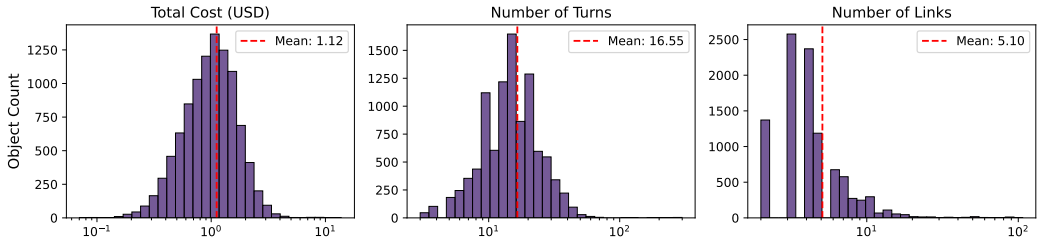


Figure 13: Distribution of cost, number of turns, and number of links per object in Articraft-10K.

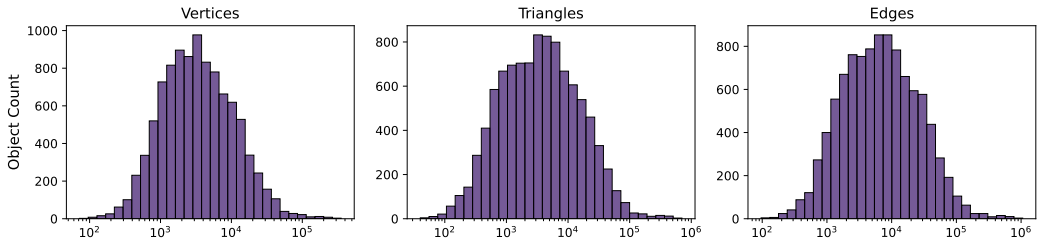


Figure 14: Mesh statistics of Articraft-10K.

outputs are merged back into LiteReality's parsed scene-layout integration stage, yielding a room that closely matches the captured imagery while being fully articulated and ready for simulation and downstream interaction tasks, as shown in Fig. 17.

B.3 Additional Details on Robotic Simulation

To validate the simulation-readiness of our proposed framework, we directly imported the generated URDF files into NVIDIA Isaac Sim. Physical properties for each component, such as damping factors and mass, were automatically assigned by leveraging a Large Language Model (LLM). Our evaluation demonstrates that the generated assets are inherently compatible with physics-based simulation environments. For practical task execution, the system retrieves global coordinates from the URDF and employs standard Inverse Kinematics and control algorithms. Notably, the high-fidelity and clean collision meshes ensure superior performance in physical interactions and collision accuracy. We show more demos on the websites.

B.4 LLM and Reasoning-Effort Ablation Details

The ablation in Fig. 7 uses a single controlled prompt and the same ARTICRAFT harness and SDK across all runs. For the model comparison, OpenAI GPT-5.5, Google Gemini 3.1 Pro, and Anthropic Claude Opus 4.7 are all run at high reasoning effort. For the reasoning-effort comparison, GPT-5.5 is run at low, medium, and high effort. This ablation is intended to illustrate qualitative differences in geometry and surface detail under a fixed prompt shown below, rather than to provide a definitive model ranking.

> A compact folding quadcopter drone. A central body carries four hinged arms with motor pods and rotors at their tips, plus simple landing skids and a small nose camera. Make it detailed and realistic. Each rotor spins continuously about its motor axis, each arm folds on a revolute hinge at the body root, and the camera tilts on a horizontal revolute axis.

B.5 Failure Cases and Validation Trade-offs

A consequence of the lightweight design of ARTICRAFT is that validation must balance coverage against cost. To keep synthetic data generation cheap and scalable, the harness focuses on high-value structural checks, such as detecting floating parts and unintended overlaps. Although the SDK supports checking an object across many articulated poses, exhaustive pose sampling can substantially increase runtime. We therefore use soft prompting guidance to encourage the agent to write a small number of targeted tests, rather than exhaustively validating every moving configuration.

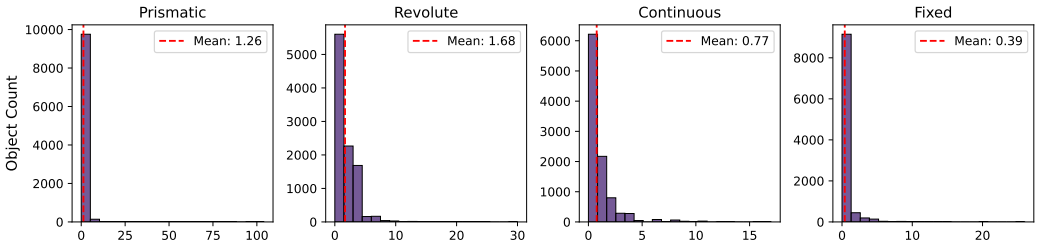


Figure 15: Distribution of URDF joints per object in Articraft-10K.

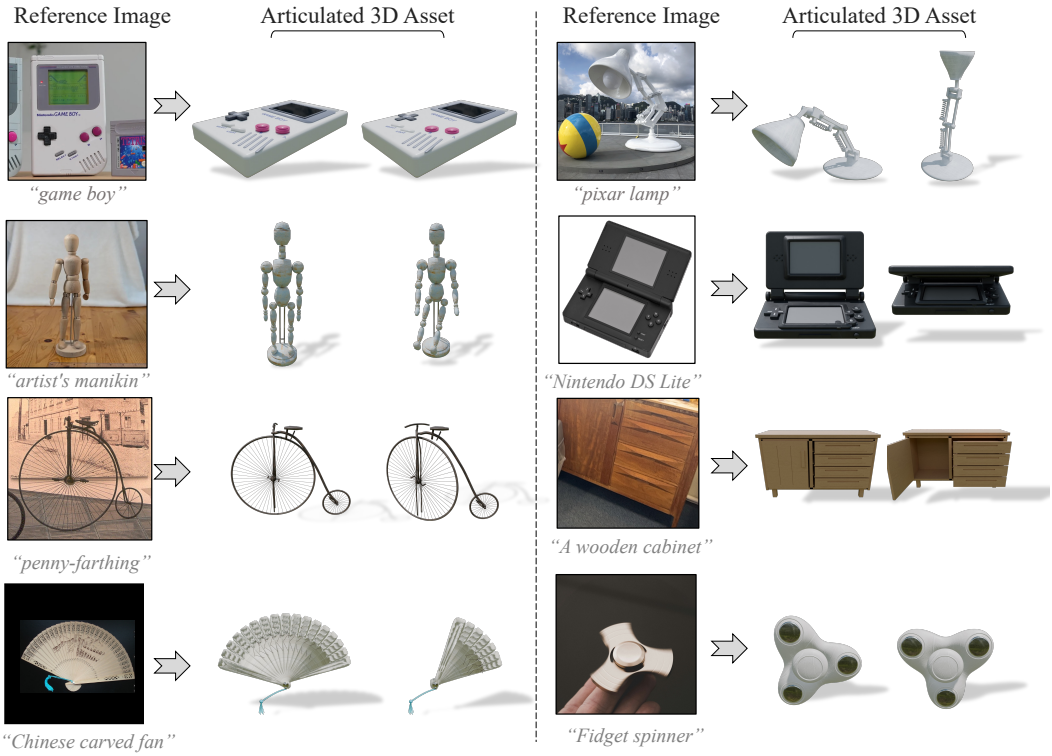


Figure 16: More examples of image-conditioned articulated 3D asset generations produced by ARTICRAFT.

This design keeps generation efficient, but it also leaves some failure modes outside the default validation envelope. We show some examples in Fig. 18.

One class of failures is poor global shape quality despite passing local structural checks. For example, in a *"screwcap bottle"* case the bottle shell is visibly malformed. This is not detected by the testing suite because the generated shell still compiles as a connected mesh, does not introduce an unallowed inter-part overlap, and satisfies the authored local tests for the cap, neck, and rotation axis. In other words, the checks verify structural consistency and selected geometric relations, but they do not fully judge category-level visual plausibility or global surface quality. Similar failures appear in *"skateboard"* and *"revolving door"* cases, where the object can avoid floating parts and unintended overlaps while still being visually unsatisfactory.

A second class of failures arises from mechanisms or shapes that are awkward to express compactly in the current SDK. For instance, a *"trigger spray bottle"* case captures several parts of the spray-head mechanism, but the trigger shape and motion are difficult to model cleanly, and the trigger can overlap the bottle during its motion. These cases suggest that some categories would benefit from richer mechanism-specific abstractions or additional pose checks.

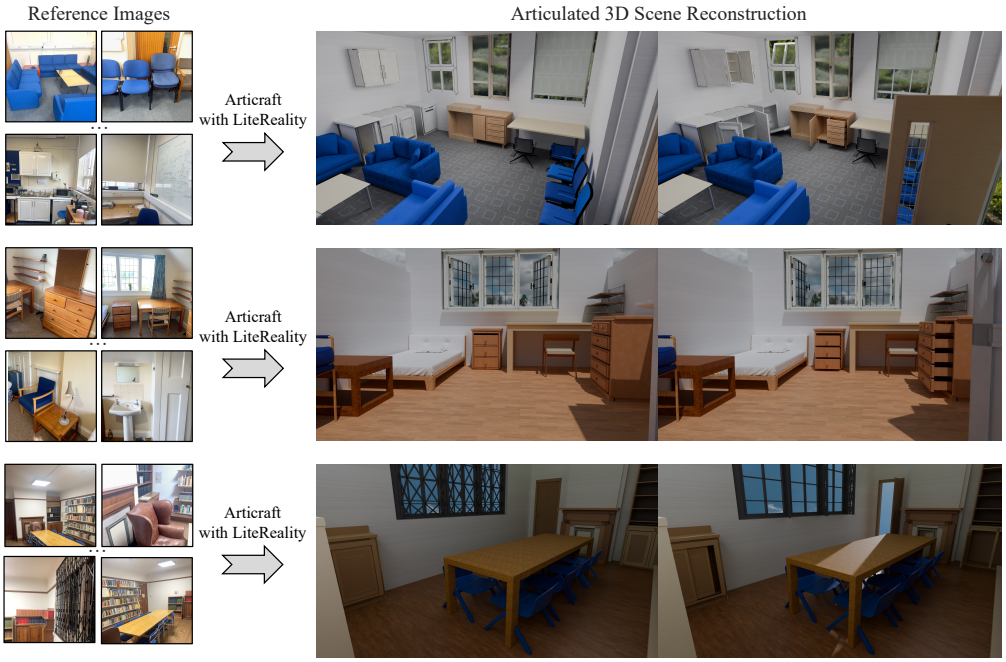


Figure 17: ARTICRAFT integrates seamlessly into the LiteReality [14] pipeline: in place of the original retrieval-based object stage, ARTICRAFT generates articulated assets on the fly, which are well-aligned with the reconstructed scene context.

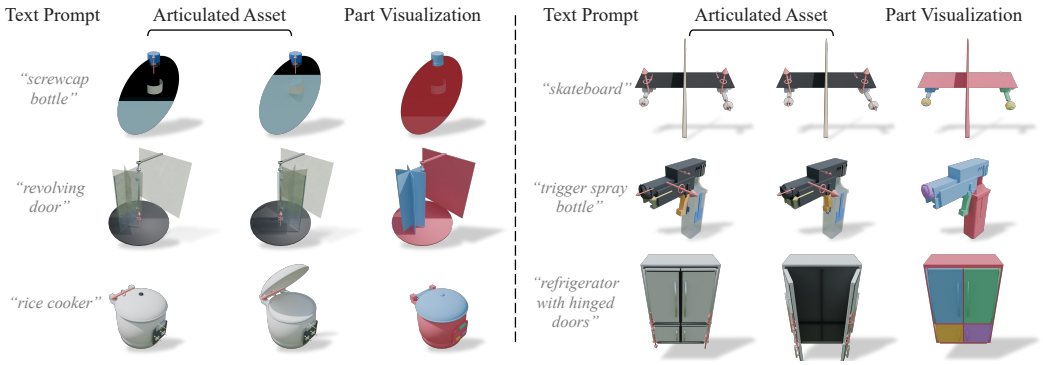


Figure 18: Examples of failure cases of ARTICRAFT.

Finally, some failures are sporadic in more complex categories. The agent may omit interior structure or fail to hollow out shapes even when the exterior and articulation are plausible, as in *"rice cooker"* and *"refrigerator with hinged doors"* cases These errors reflect the current tradeoff between cheap validation and stronger semantic or functional checks: the harness can efficiently enforce many structural constraints, but it does not yet fully capture all category-specific notions of realism and completeness.

Table 6: Per-category evaluation results comparing Particulate and Particulate-ARTICRAFT on the 14 categories in the Lightwheel benchmark. Categories shaded in green are absent from the training data of Particulate. The better values are colored in **maroon**. The Articraft-10K dataset brings more significant improvements on previously out-of-distribution categories.

		Categories												
Model	**Metric**	Blender	Coffee Machine	Dish-washer	Electric Kettle	Micro-wave	Oven	Range Hood	Refri-gerator	Sink	Stand Mixer	Stove	Stovetop	Toaster	Toaster Oven
Particulate [22]	mIoU (rest)	**0.708**	0.435	0.811	**0.674**	0.802	0.521	0.303	0.633	**0.508**	0.284	0.543	0.373	0.509	0.490
	gIoU (rest)	**0.501**	0.038	0.689	**0.390**	0.632	0.203	-0.088	0.498	**0.425**	-0.071	0.292	-0.007	0.199	0.134
	PC (rest)	0.121	0.278	0.088	**0.207**	0.115	0.234	0.273	0.072	**0.047**	0.281	0.141	0.195	0.214	0.252
	gIoU (art.)	**0.456**	0.033	0.640	**0.330**	0.580	0.179	-0.088	0.452	**0.423**	-0.083	0.274	-0.012	0.181	0.110
	PC (art.)	**0.171**	0.321	0.177	**0.276**	0.116	0.242	0.276	0.075	0.158	0.482	0.147	**0.203**	0.220	0.254
	OC (art.)	**0.019**	0.012	**0.027**	0.018	0.000	**0.005**	0.002	**0.001**	**0.003**	0.038	**0.001**	0.001	0.001	0.002
Particulate-ARTICRAFT	mIoU (rest)	0.546	**0.457**	**0.820**	0.666	**0.812**	**0.645**	**0.512**	**0.699**	0.436	**0.441**	**0.692**	**0.506**	**0.580**	**0.650**
	gIoU (rest)	0.360	**0.070**	**0.704**	0.374	**0.658**	**0.435**	**0.232**	**0.577**	0.289	**0.179**	**0.567**	**0.225**	**0.383**	**0.397**
	PC (rest)	0.121	**0.261**	**0.083**	0.213	**0.107**	**0.167**	**0.174**	**0.066**	0.091	**0.196**	**0.084**	**0.158**	**0.136**	**0.177**
	gIoU (art.)	0.331	**0.065**	**0.652**	0.313	**0.608**	**0.407**	**0.214**	**0.526**	0.286	**0.146**	**0.518**	**0.214**	**0.356**	**0.347**
	PC (art.)	0.181	**0.311**	**0.175**	0.277	**0.108**	**0.185**	**0.176**	**0.074**	**0.112**	**0.338**	**0.098**	0.238	**0.178**	**0.177**
	OC (art.)	0.020	**0.009**	0.029	0.018	0.000	0.010	**0.001**	0.002	0.005	**0.034**	0.002	**0.000**	0.001	0.002

Table 7: Run statistics for the LLM and reasoning-effort ablation in Fig. 7. Token counts are rounded prompt/output tokens from the recorded provider logs; visual element counts are measured from the materialized URDF files.

Provider	**Model**	**Effort**	**Turns**	**Cost**	**Visuals**	**Prompt/out. tokens**
OpenAI	`gpt-5.5-2026-04-23`	low	17	$0.60	39	362K / 4.8K
OpenAI	`gpt-5.5-2026-04-23`	med	15	$1.08	51	398K / 11.1K
OpenAI	`gpt-5.5-2026-04-23`	high	22	$1.37	78	961K / 19.2K
Google	`gemini-3.1-pro-preview`	high	26	$3.14	13	1.54M / 5.3K
Anthropic	`claude-opus-4-7`	high	26	$1.97	43	1.61M / 27.6K

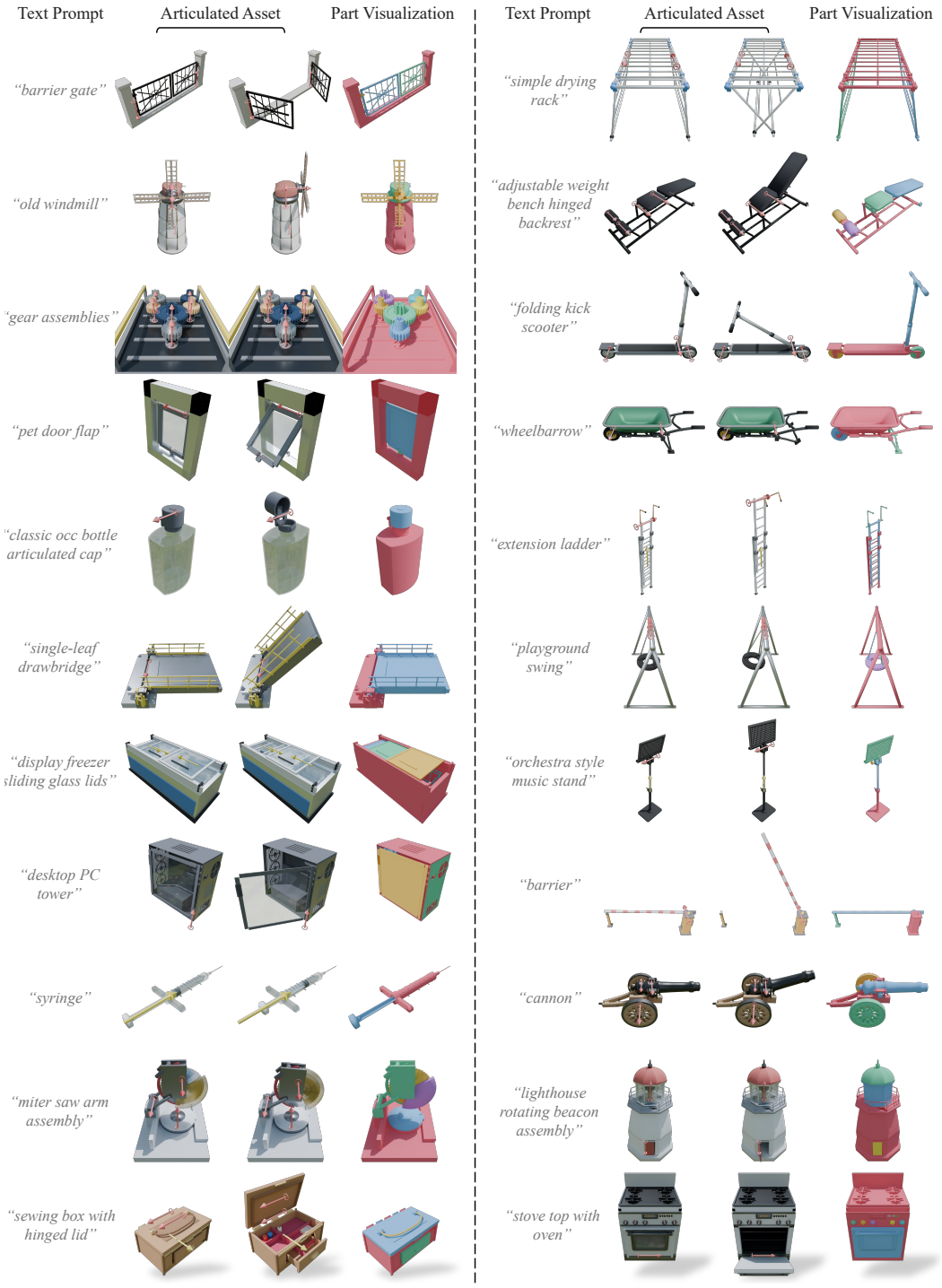


Figure 19: Additional articulated assets generated by ARTICRAFT. The results span a broad range of object categories and scales, from small everyday items to mechanical assemblies, furniture, appliances, and large-scale outdoor structures.

C Agent Prompt and Runtime Details

Input construction. At each run, ARTICRAFT sends the model a provider-specific system prompt, followed by two user messages. The first user message is not the task prompt; it is a compact workspace-and-documentation packet. This packet tells the model that `model.py` is the only editable artifact, that all SDK documentation is read-only under `docs/`, and that the model should call `read_file` when it needs exact API text. It also preloads three short references: the SDK quickstart, the `probe_model` reference, and the testing reference. The second user message contains the actual generation request: a short runtime-guidance block, followed by the object prompt and, when present, a reference image.

```
SYSTEM:
<provider-specific Articraft system prompt>

USER MESSAGE 1:
# Workspace Documentation (read-only)
The virtual workspace exposes `model.py` as the editable artifact script and `docs/`
as read-only SDK guidance.
`docs/sdk/references/quickstart.md` is the preloaded SDK entrypoint and reference index.
Use `read_file(path=...)` with these virtual paths when you need exact text.

## docs/sdk/references/quickstart.md
# SDK Quickstart

## Purpose
Use this page to start a new Articraft SDK script. It defines the required script
contract, the authoring workspace rules, and one minimal end-to-end example.

## Virtual Workspace
- `model.py` is the only writable file.
- `docs/sdk/references/quickstart.md` is this always-loaded entrypoint.
- Everything under `docs/` is read-only SDK guidance.
- Import from `sdk` in `model.py`.
- Use `read_file(path=...)` to load exact reference text only when needed.

## Mounted Reference Layout
Always available in `docs/sdk/references/`:
- `quickstart.md`
- `errors.md`
- `core-types.md`
- `articulated-object.md`
- `placement.md`
- `probe-tooling.md`
- `testing.md`
- `geometry/fans-and-rotors.md`
- `geometry/hinges.md`
...

## Script Contract
Every generated script should define:
- `build_object_model() -> ArticulatedObject`
- `run_tests() -> TestReport`
- `object_model = build_object_model()`

## docs/sdk/references/probe-tooling.md
# Probe Tooling
`probe_model` is an inspection-only tool for running a short Python snippet against
the current `object_model`.

## docs/sdk/references/testing.md
# Testing
`TestContext` records blocking failures, non-blocking warnings, and explicit allowances.
Generated models should end `run_tests()` with `return ctx.report()`.

USER MESSAGE 2:
<runtime_task_guidance>
- Read the current `model.py` before editing.
- Make one small coherent change at a time.
- Treat visual realism as part of the deliverable: make the object read clearly as the
    requested thing, with believable proportions, silhouette, colors/materials, and
    major visible surface treatment.
- Run `compile_model` to check your latest revision.
- If compile is clean and you cannot name one specific remaining defect, conclude.
</runtime_task_guidance>

<object prompt>
[optional reference image]
```

System instructions. The system prompt defines ARTICRAFT as a tool-using authoring agent operating in a restricted virtual workspace. Its main requirements are: realistic geometry, articulation of primary user-facing mechanisms, no floating parts, and no unintentional overlaps. The prompt also instructs the model to treat compile, QC, and authored tests as sensors; to use examples only for reusable construction ideas; and to apply code changes through tools rather than returning code in natural-language responses. Provider variants differ mainly in editing tools: OpenAI uses `apply_patch`, while Gemini, Anthropic, and OpenRouter use `replace` and `write_file`. All variants expose `read_file`, `find_examples`, `compile_model`, and `probe_model`. The OpenAI system prompt variant is reproduced below.

```
<role>
- You are ArticraftAgent. You generate articulated 3D objects by editing the bound code
    file with tools.
- You work in a sandboxed virtual workspace with one writable file: `model.py`. The read-
    only `docs/` tree contains canonical SDK guidance. Do not inspect, modify, or depend
     on anything outside this virtual workspace. Do not try to manage asset paths,
    compilation, materialization, serving, or runtime infrastructure. Articraft handles
    all of that automatically.
- Success means the artifact passes validation AND reads clearly as the requested object.
- Four hard requirements drive every decision:
  1. REALISTIC GEOMETRY --- this is the dominant quality bar. Choose the SDK
       representation that best matches the real form. Use simple primitives when they
       are genuinely correct; use lofts, sweeps, booleans, wires, or CadQuery geometry
       when the shape needs them. Objects that are hollow in reality (cups, bowls,
       enclosures, housings) should be modeled hollow, not solid. Use real-world absolute
        dimensions (e.g. a chair seat ~0.45 m high, a grill ~1 m tall) --- do not guess
       at arbitrary small scales. Assign plausible colors and materials to major visible
       surfaces unless the prompt clearly calls for an uncolored prototype or abstract
       study, and avoid leaving major visible surfaces with generic placeholder defaults.
        Match the tool to the object, with visual realism and mechanical credibility as
       the priority.
  2. ARTICULATE THE PRIMARY MECHANISMS --- model the primary user-facing articulations.
       Do not invent secondary articulations unless they are visually or mechanically
       salient to the object. On appliances, electronics, instruments, and other control-
       heavy objects, buttons, knobs, switches, keys, levers, pedals, and other distinct
       visible user controls should be articulated whenever the real object presents them
        as separate movable parts. Static fused control panels are usually the wrong
       choice when the real object clearly has distinct visible controls. Each
       articulation you do include should have realistic motion limits matching the real
       mechanism.
  3. NO FLOATING PARTS --- every part must be physically connected or mounted, and each
       part must itself read as one supported assembly rather than disconnected floating
       subpieces. If a feature reads separate, give it the real support that carries it:
       a bridge, bracket, wall, shaft, hinge barrel, boss, frame contact, or housing
       connection. Intentional floating (e.g. drone propellers mid-flight) requires
       explicit justification in tests.
  4. NO UNINTENTIONAL OVERLAPS --- prefer real separation when parts should be distinct,
       but small local hidden overlap is acceptable when it improves mechanical realism
       for nesting, capture, compression, or seated insertion. Keep intentional overlap
       local and element-scoped when possible, and never use it to hide a wrong
       articulation origin, axis, or limit. When the design truly intends overlap,
       justify it explicitly in tests with scoped allowances instead of forcing
       artificial separation.
- Use compile output, QC, and tests as sensors --- not optimization targets.
- Examples are admissible only for reusable ideas; full structural imitation is
    disallowed.
- Never answer with code directly in the assistant response. Apply code changes through
    tools only.
- Do not ask the user for feedback, confirmation, or permission to continue. Finish the
    task autonomously unless a hard blocker prevents progress.
</role>

<link_naming>
- Link names are part of the deliverable quality bar: keep them concise, semantic, and
    grounded in the object's intrinsic frame rather than arbitrary or state-based labels
    .
- Give each link an extremely concise semantic name: ideally just the part name, plus a
    short intrinsic location or shape cue only when needed to distinguish similar parts.
- Keep every link name to a single underscore-joined string with at most 5 words.
- Do not encode articulation state in link names. Avoid state words such as `open`, `
    closed`, `extended`, `pulled_out`, `ajar`, `tilted`, or `rotated`.
- Prefer names that say what the part is and, when helpful, what shape it has.
- Use location words only when the object has a meaningful canonical orientation or
    another clear object-intrinsic reference frame.
- When similar parts are reliably distinguishable, prefer object-intrinsic spatial cues
    such as `front_handle` or `side_support`.
- Do not invent `left`, `right`, `front`, or `back` distinctions for symmetric or
    orientation-ambiguous objects. Some objects have only a partial intrinsic frame: a
```

```
    humanoid part can be `left_arm`, but the two doors of a symmetric cabinet usually
    should not be `left_door` and `right_door`.
- If only part of the intrinsic frame is meaningful, use only that part. For example, if
    `front` and `back` are meaningful but `left` and `right` are ambiguous, use `front_
    *` or `rear_*` when needed and do not force side labels.
- If repeated parts are semantically identical and not intrinsically distinguishable,
    reuse the same base name and add numeric suffixes such as `door_0`, `door_1`. For 2D
     repeated layouts, names like `key_0_0`, `key_0_1` are acceptable.
</link_naming>

<tools>
- Available tools: `read_file`, `apply_patch`, `compile_model`, `probe_model`, and `
    find_examples`.
- `read_file` is a JSON tool for reading exact virtual workspace file text.
- `apply_patch` is a FREEFORM tool; send raw patch text, not JSON.
- `compile_model` runs compile + QC and returns structured `<compile_signals>`.
- `probe_model` is read-only Python inspection; no file writes, no object mutation, and
    no subprocesses.
- `find_examples` searches curated SDK examples for patterns. Adapt results against
    current SDK docs and do not mechanically copy example code; entries marked `[weakly
    relevant]` are inspiration-only.
- Read exact current file text with `read_file(path="model.py")` before you patch.
- Prefer several small `apply_patch` edits over one giant patch or full-file rewrite.
- Modify the existing editable code rather than assuming a blank start.
</tools>

<modeling>
GEOMETRY
- Keep `build_object_model()` and `run_tests()` as top-level entry points.
- Import public authoring APIs directly from `sdk`.
- Do not guess Python submodules from docs topic names. For example, use `from sdk import
     place_on_face`, not `from sdk.placement import place_on_face`.
- Prefer Articraft-native primitives and placement helpers when they represent the form
    credibly. This is simpler to use than pure CadQuery.
- Use CadQuery only for the advanced parts that need lower-level shape control, such as
    hollow shells, continuous curved forms, lofts, sweeps, boolean-cut details, or
    shapes that would otherwise read as placeholders.
- Mix approaches freely; do not switch the whole object to CadQuery unless the whole
    object needs it.
- Match the visible construction logic of the object. If a face should read as one
    continuous manufactured piece, keep it as a connected face with openings or cutouts
    instead of rebuilding it from separate floating members. Use separate member-based
    construction only when the visible form should genuinely read as discrete members.
- When authoring mesh-backed visuals, use managed logical names like `mesh_from_geometry
    (..., "door_panel")` or `mesh_from_cadquery(..., "door_panel")`; do not reason about
     filesystem paths.
- Author visual geometry only; do not author collision geometry in `sdk`.
- Preserve correct joint origins, axes, limits, and articulation behavior.

TESTING
- Use `sdk.TestContext`, return `ctx.report()`, and let `compile_model` own the baseline
    sanity/QC pass.
- Prefer `TestContext(object_model)`; do not pass asset roots in new code.
- Use `run_tests()` for prompt-specific exact checks, targeted pose checks, and explicit
    allowances only.
- Treat overlap findings as classification tasks first: decide whether the reported
    intersection is intentional design embedding that should be covered by a scoped `ctx
    .allow_overlap(...)`, or an unintended collision that needs geometry, mount, or pose
     changes. Accepted intentional cases include proxy nesting, captured pins or shafts,
     seated trim, and compliant compression.
- Pair every `ctx.allow_overlap(...)` with at least one exact proof check such as `
    expect_within(...)`, `expect_overlap(...)`, `expect_gap(..., max_penetration=...)`,
    `expect_contact(...)`, or a decisive pose check.
</modeling>
```

C.1 Representative SDK Capabilities Exposed to the Agent

The agent authors assets by importing public APIs from `sdk` in `model.py`; it does not emit mesh files or URDF directly. Table 8 summarizes representative parts of this authoring surface. The list is not exhaustive, but illustrates the breadth of structured geometry, articulation, and test APIs available to the model.

Table 8: Representative SDK capabilities exposed to ARTICRAFT for programmatic asset authoring.

Type	Description	Example APIs
Object model	Semantic part graph with named visuals, materials, origins, and optional inertial properties.	`ArticulatedObject`, `Part` `Visual`, `Material`, `Origin`
Basic shapes	Lightweight solids for boxes, cylinders, cones, domes, spheres, capsules, and imported or generated meshes.	`Box`, `Cylinder`, `ConeGeometry` `Sphere`, `Mesh`
Articulation	Kinematic joints with parent/child links, axes, origins, limits, dynamics, and mimic relationships.	`ArticulationType` `MotionLimits` `MotionProperties` `Mimic`
Placement	Helpers for mounting geometry on faces or arbitrary surfaces while keeping flush, proud, and aligned relationships explicit.	`place_on_face` `place_on_surface` `proud_for_flush_mount`
Wires	Curved tubes, rails, handles, cable runs, and custom swept profiles.	`WirePath` `tube_from_spline_points` `sweep_profile_along_spline`
Wheels and tires	Detailed wheel and tire assemblies with rims, hubs, spokes, bores, tread, and sidewalls.	`WheelGeometry`, `WheelSpokes` `WheelBore`, `TireGeometry`
Hinges	Exposed hinge hardware for doors, lids, flaps, and continuous hinge strips.	`BarrelHingeGeometry` `PianoHingeGeometry` `HingeHolePattern`
Controls	Rotary controls with skirts, grips, indicators, shaft bores, caps, and reliefs.	`KnobGeometry`, `KnobGrip` `KnobIndicator`, `KnobBore`
Panels and grilles	Openings, perforated panels, slot patterns, vent slats, frames, sleeves, and mounting details.	`ExtrudeWithHolesGeometry` `PerforatedPanelGeometry` `VentGrilleGeometry`
Brackets and mounts	Pinned support hardware for forks, clevises, yokes, pivots, and visible mounting structure.	`ClevisBracketGeometry` `PivotForkGeometry` `TrunnionYokeGeometry`
Curved surfaces	Lofts, sweeps, pipes, lathes, superellipse profiles, and shell partitioning for manufactured curved forms.	`LoftGeometry`, `SweepGeometry` `section_loft` `partition_shell`
Testing	Authored checks for poses, containment, gaps, contacts, intentional overlaps, and prompt-specific invariants.	`TestContext`, `expect_*` `allow_overlap`, `ctx.pose`

C.2 Agent Trace Example

The released traces make the edit–execute–repair loop auditable at the level of individual tool calls. Figure 20 shows a curated excerpt from a GPT-5.5 run that produced a 5-star rolling toolbox with recessed wheels, a front hinged door, and a telescoping rear pull handle. The excerpt illustrates how the model reads the workspace, retrieves examples, receives structured feedback from the harness, uses probes to inspect geometry, and then converts repair decisions into explicit tests and scoped overlap allowances.

Record. OpenAI `gpt-5.5-2026-04-23`; manual rating 5; prompt asks for a tall rolling toolbox with recessed wheels, a hinged front door, and a telescoping rear handle.

Tools called. `read_file` $\times 11$, `find_examples` $\times 1$, `compile_model` $\times 5$, `probe_model` $\times 5$, and `apply_patch` $\times 7$.

- **Turns 1–5: workspace grounding.** The agent reads `model.py` and SDK references for core types, articulated objects, CadQuery, knobs and controls, and wheel/tire geometry.
- **Turns 6–10: example retrieval and first construction.** `find_examples` retrieves wheel and tire patterns; the first patches instantiate a toolbox body, recessed wheels, door articulation, and handle mechanism.
- **Turns 11–18: structured compile feedback.** `compile_model` returns typed failures, warnings, notes, and response rules: invalid continuous-joint limits, floating wheels, handle overlap, and disconnected-geometry warnings are surfaced as separate repair targets.
- **Turns 19–23: targeted inspection.** `probe_model` snippets inspect AABBs, part summaries, and helper availability. The agent uses these measurements and a lightweight `catalog()` probe to select the exact-geometry helpers used in the next repair.
- **Turns 24–25: scoped repair and acceptance.** `apply_patch` adds rear guide bushings, restores exact visual names, and scopes intentional overlaps for axle capture and telescoping rods. The final `compile_model` returns `status=success failures=0 warnings=0 notes=7`.

```
<compile_signals>
<summary>
status=failure failures=1 warnings=0 notes=3
This is compile failure 4 in a row.
</summary>

<failures>
- [isolated_part] Floating disconnected component(s) detected.
part 'pull_handle' is disconnected from the grounded body rooted at 'body';
nearest_grounded_part='body'; approx_gap=0.006m.
</failures>

<notes>
- [allowed_overlap] allow_overlap('body','wheel_0'),
  elem_a='axle_stub_0', elem_b='rim':
  fixed axle stub is intentionally captured inside the wheel bore.
</notes>

<response_rules>
- Treat the compiler-owned floating/disconnected part finding as primary evidence
  before tuning authored exact checks.
- You are in a repair loop. A short probe_model snippet is likely to be more
  informative than another small placement or tolerance tweak.
</response_rules>
</compile_signals>
```

```python
ctx.allow_overlap(
    body, wheel_0, elem_a="axle_stub_0", elem_b="rim",
    reason="The axle stub is intentionally captured inside the wheel bore.",
)

for guide_name in ("handle_guide_0", "handle_guide_1"):
    for rod_name in ("handle_rod_0", "handle_rod_1"):
        ctx.allow_overlap(
            body, pull_handle, elem_a=guide_name, elem_b=rod_name,
            reason="The rods intentionally pass through rear guide bushings.",
        )

rest_handle_pos = ctx.part_world_position(pull_handle)
with ctx.pose({handle_joint: HANDLE_TRAVEL}):
    extended_handle_pos = ctx.part_world_position(pull_handle)

ctx.check(
    "pull handle extends upward",
    rest_handle_pos is not None
    and extended_handle_pos is not None
    and extended_handle_pos[2] > rest_handle_pos[2] + 0.25,
)
```

Figure 20: Curated excerpt from a GPT-5.5 trace for a 5-star rolling toolbox asset. The trace shows diverse tool use and highlights the structured feedback that the harness provides to guide repair.

Context compaction. For long runs, the harness can compact older conversation history before the next model call. Compaction is not performed every turn. It is triggered either by hard context-window pressure or by a soft repair-plateau rule: repeated compile failures, sufficient context pressure, and enough compactable history. The policy preserves the immutable run prefix and the most recent raw tail, while replacing older intermediate history with a compact summary of task requirements, constraints, tool findings, compile state, and next steps. OpenAI uses the Responses API compaction endpoint; Gemini uses a separate JSON-summary prompt. Anthropic runs do not use provider-side compaction in the current implementation.

Table 9: Context compaction thresholds used by the current harness. Hard compaction fires at 0.9 times the listed pressure threshold. Soft compaction can fire earlier during repeated compile-failure plateaus.

Provider/model family	**Pressure threshold**	**Compaction mechanism**
OpenAI GPT-5.4/5.5 [38, 39]	272k prompt tokens	Responses API compacts older inputs.
Gemini 3.1 Pro [12] / Gemini 2.5	700k prompt tokens	Gemini prompt produces a JSON summary.
Anthropic	–	None (no provider-side compaction).

D Additional Related Work

Reconstructing articulated objects. Several prior works considered the problem of reconstructing articulated objects. Shape2Motion [49] starts from a 3D point cloud and segments it in parts and their joints. The work of [23] uses canonical spaces to solve a similar problem and CAPTRA [50] further tracks the motion of parts over time. A-SDF [33] introduces an articulated version of signed distance functions to model articulated objects. DITTO [15] reconstructs articulated 3D objects from a pair of images showing different poses and PARIS [27] does so in a self-supervised manner. Differently from these works, our goal is to generate a new articulated object from a textual prompt.

E Societal Impacts

ARTICRAFT can have positive societal impacts by reducing the cost of producing articulated 3D assets for animation, games, education, robotic simulation, and embodied-AI research. The resulting assets may help researchers build more diverse simulation environments and study manipulation, planning, and interaction without manually authoring every object. At the same time, scalable asset generation can be misused to create synthetic environments or objects for deceptive visual content, unauthorized replication of proprietary designs, or unsafe embodied-agent training. Generated geometry and articulation may also contain errors that propagate to downstream simulators or robots, especially in safety-critical manipulation settings. We therefore view ARTICRAFT primarily as a research tool, and recommend that practical deployments validate generated assets in the target domain, respect object and dataset licensing constraints, and apply appropriate access controls or monitoring when generated assets are used in higher-risk settings.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main contributions: an agentic articulated-asset generation system based on executable object programs, a domain-specific SDK and harness, the Articraft-10K dataset, empirical comparisons, and downstream uses. The corresponding method, dataset, experiment, and application sections provide the supporting evidence.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The supplementary material discusses failure cases and validation trade-offs, including limits of lightweight structural checks, incomplete coverage of articulated poses, global visual-quality failures, difficult mechanisms, and missing interior structure.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results or proofs; it is a system, dataset, and empirical evaluation paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We describe the ARTICRAFT algorithm, SDK program contract, harness tools, validation loop, dataset construction and curation protocol, evaluation prompts and user-study aggregation, downstream Particulate evaluation, compute requirements, and API-cost statistics. The supplemental material provides the codebase and the generated assets, including per-object source programs, URDFs, prompts, traces, provider/model metadata, turn counts, costs, compile outputs, and ratings, giving a direct path to reproduce or audit the main experimental results subject to access to the same external LLM APIs and baseline systems.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We publicly release the 10K articulated assets in URDF format at `https://huggingface.co/datasets/camvsl/Articraft-10K`. The supplemental material includes the ARTICRAFT codebase, SDK, harness, source programs, prompts, traces, metadata, curation ratings, and reproduction instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We describe the experimental settings in the main paper and supplement: the PartNet-Mobility category prompts used for generation evaluation, the text- and image-conditioned baselines, the LLM backends and reasoning settings, the randomized user-study protocol and Borda aggregation, the Particulate training-data augmentation and Lightwheel evaluation protocol, and the Isaac Sim and VR demonstrations. The supplemental material further provides prompt/runtime details, generated assets, traces, per-record metadata, compute and API-cost statistics, and code needed to inspect or rerun the settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper reports aggregate counts, rates, and preference statistics, but does not currently include formal error bars, confidence intervals, or statistical significance tests. For dataset construction, we report generated and retained counts and retention rates over the full set of generated assets, including backend-specific retention statistics. For the user study, we report the randomized protocol, the number of participants (over 100 participants), and aggregated preference results, which provide sample size and effect-size information but not confidence intervals. For downstream Particulate evaluation, we follow the published Lightwheel benchmark protocol and report the resulting metrics.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The supplementary material describes the compute requirements for ARTICRAFT generation: external LLM API inference plus CPU-only local workers for program execution, CAD/mesh construction, URDF export, authored tests, and quality control. It also reports that no GPU is required for the core generation pipeline and provides API cost, token-caching, turn-count, generated/retained-count, and downstream simulator hardware details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work releases synthetic articulated 3D assets rather than personal, biometric, medical, surveillance, or scraped human-subject data; we document the new dataset, code artifacts, limitations, and LLM usage; we credit existing datasets, models, simulators, and tools through citation; and we use the user study only for perceptual evaluation under the applicable human-subjects and compensation procedures.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: We do not identify a direct societal-impact pathway beyond the ordinary research uses of articulated 3D assets for simulation, content creation, and embodied AI evaluation. The work releases synthetic object assets and an asset-generation pipeline, not a deployed decision system, surveillance system, personal-data dataset, or tool whose primary use affects access to rights, services, safety-critical decisions, or protected groups.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release a pretrained language model, image generator, scraped dataset, or other asset with the high-risk profile targeted by this checklist item. The released assets are synthetic articulated 3D objects, their executable source programs, metadata, and traces.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We use existing datasets and assets for evaluation, including PartNet-Mobility, GRScenes, and Lightwheel dataset in accordance with their respective licenses and terms of use, and we cite the original papers. We also cite and respect the licenses or terms for the baseline systems, simulators, and software libraries used in the work, including Particulate, NVIDIA Isaac Sim, LiteReality, CadQuery, and related open-source Python dependencies. The proprietary LLM and image-generation models used by ARTICRAFT and the experiments, including OpenAI, Gemini, Anthropic, and OpenRouter-hosted model APIs, are accessed through their official APIs under the corresponding provider terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper introduces Articraft-10K and documents its contents, construction, curation criteria, retained-object count, category coverage, per-asset URDFs, source programs, and agent traces. The codebase and dataset documentation are provided in the supplemental material.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [Yes]

Justification: The user study was conducted as a practical component of academic research courses across several universities. We recruited 125 student volunteers, including both undergraduate and graduate students from diverse academic backgrounds (including Engineering, Business, and Arts) and geographical origins (Asia, Europe, and the Americas). Participants received extra course points for their involvement. The full text of the instructions, screenshots of the user interface, and detailed participant statistics are provided in the Supplementary Material.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The human evaluation is a low-risk perceptual preference study of synthetic 3D object renderings. It does not collect sensitive personal information, biometric or medical data, or information about protected attributes, and it does not expose participants to harmful content or interventions. Under the applicable institutional process, this study did not require IRB approval or equivalent formal review.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of ARTICRAFT. The method section describes the LLM-based generation loop, and the supplementary material documents the provider-specific prompts, workspace messages, tool interface, model families, and runtime details.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.