

# Lucid-XR: An Extended-Reality Data Engine for Robotic Manipulation

Anonymous Author(s)

Affiliation

Address

email

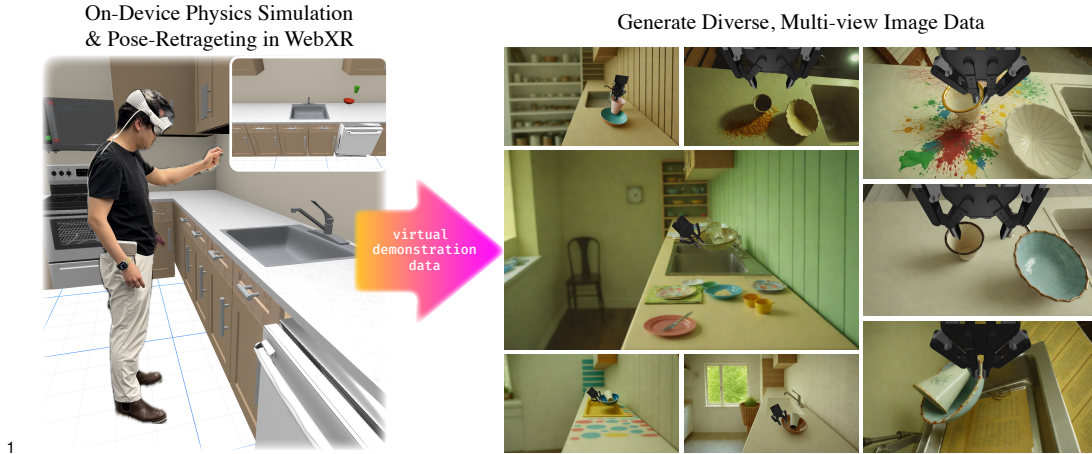


Figure 1: **An Extended Reality Data Engine for Robotic Manipulation.** *Left:* we deliver physics simulation to run directly on the XR devices via the web browser, to enable internet-scale crowdsourcing of demonstration data collection. *Right:* Our GenAI-powered synthetic data engine creates steerable, diverse, and realistic multi-view visual data to train real-world robots.

**Abstract:** We introduce Lucid-XR, a generative data engine for creating diverse and realistic-looking multimodal data to train real-world robotic systems. At the core of Lucid-XR is vuer, a web-based physics simulation environment that runs directly on the XR headset, enabling internet-scale access to immersive, latency-free virtual interactions without requiring specialized equipment. The complete system integrates on-device physics simulation with human-to-robot pose retargeting. Data collected is further amplified by a physics-guided video generation pipeline commandable via natural language specifications. We demonstrate zero-shot transfer of robot visual policies to unseen, cluttered, and badly lit evaluation environments, after training entirely on Lucid-XR’s synthetic data. We include examples across bimanual and dexterous manipulation tasks that involve soft materials, air dynamics, loosely bound particles, as well as rigid body contact. Project website: <https://lucidxr.github.io>

**Keywords:** Extended-reality, world-model, synthetic data

## 1 Introduction: From Atoms to Bits

When viewed from afar, training a robot controller is not too different from making a movie, in that both involve carefully curating content. Feature films over the years have transitioned steadily from practical special effects that occur in the physical world to digitally created virtual special effects,

## Lucid-XR: An Extended-Reality Data Engine

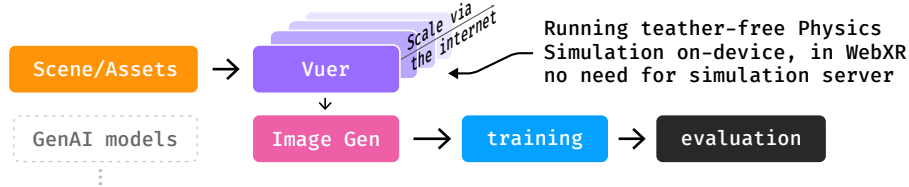


Figure 2: **System Schematic of the Lucid-XR Pipeline.** The results in this paper require hand-crafted, but basic 3D scenes. The data collection is done collectively by the authors. The resulting simulated datasets are augmented by a generative pipeline powered by language and text-to-image models.

driven primarily by creators’ demand for even greater freedom in storytelling. Hardening robotic systems for real-world deployment requires a similar level of control over the training environment to cover rare but mission-critical events that, by definition, are scarce in real-world datasets. What makes robotics more difficult is that digitally creating millions of virtual worlds at the scale required for our robots to generalize is infeasibly expensive.

In this work, we introduce Lucid-XR, an extended-reality (XR) data engine for robot manipulation that scales with the internet. Key to our vision is to deploy fast, multi-physics simulations through the internet browser to any extended reality device (XR). The goal is to enable the crowdsourcing of unlimited human demonstration data. Furthermore, virtual demonstrations in sparsely populated 3D environments alone are insufficient for training real-world computer vision systems. We leverage language and text-to-image generative models to construct a physics-guided video generation pipeline that amplifies a small number of simple designs into millions of diverse and realistic-looking multi-view images for the robot.

One challenge that makes crowd-sourcing teleoperation difficult is that retargeting human poses to robot form factors that are kinematically different involves writing custom computer code for each robot and setting up a server. We solve this problem by leveraging MuJoCo’s inverse kinematics solver and allowing users to define bindings between motion capture (MOCAP) sites and robot parts in the markup schema. This general scheme can extend across different robot form factors, enabling on-device re-targeting without requiring custom computer code.

Our contributions are three-fold: first, moving physics simulation onto the XR device, to deliver a latency-free multi-physics simulation in an immersive environment via the open internet; second, a way to retarget human pose data to virtual robots, without requiring a custom computer program. This setup also comes with a novel interaction pattern called “hitchiking controllers”, for operating virtual robot grippers at a distance in XR; and third, a demonstration of a policy trained on Lucid-XR’s synthetic data deployed on scanned digital twins of real environments.

Lucid-XR presents a viable GenAI-powered extended reality setup for robot learning and a playground for developing intuitive, software-defined human-robot interactions. We believe that lowering the barrier to entry will lead to greater realism in these virtual worlds, as well as more robust robots.

## 2 A Touch of Physics in Extended Reality (XR)

Modern internet browsers are de facto operating systems built according to a commonly agreed-upon programming interface. The key limitation is the constraint on compute and memory, as the device’s operating system needs to throttle the browser process to maintain a smooth user experience while handling a multitude of system and application tasks. In the past decade, however, the competition between vendors has significantly improved browser performance; on newer extended-reality devices, the browser receives first-class support in features sets, system resource limits, and performance [1].



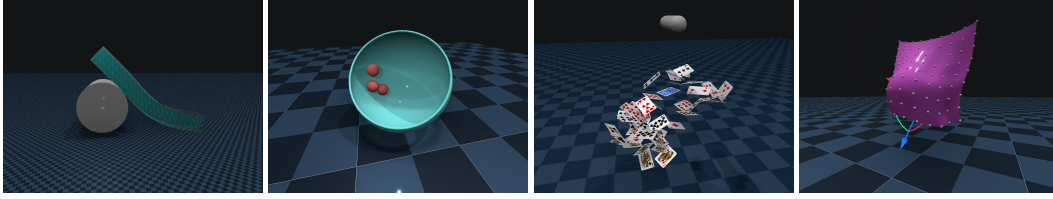


Figure 3: **Multiple types of physics running natively inside the browser in real-time.** From left to right: flexible material interacting with a solid; Signed-distance function (SDF) based collision solver for non-convex shapes; A deck of cards interacting with air/wind; Soft skin material interacting with a solid. Rendering and simulation are both native in a web browser.

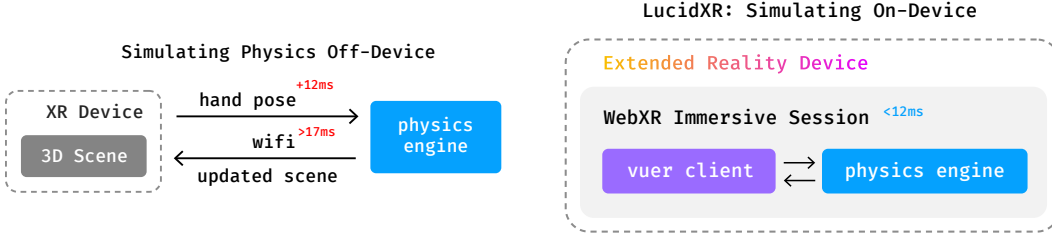


Figure 4: **Moving physics simulation on-device enables untethered access to immersive simulations.** The key benefits are twofold: first, it enables the simulation of deformable objects that involve modifying a large amount of mesh data that are too slow to send over wifi. Second, it *eliminates* delays due to network latency, allowing the simulation to run at the device’s native frame rate.

Lucid-XR leveraged three web standards. First, a simulator needs to be *fast*. We bypassed the performance and single-thread limit of the V8 javascript engine, by compiling the open-source physics simulator MuJoCo into WebAssembly. This enables us to run multi-physics simulations at close to native speed directly on the XR device within its power limit. Second, we built upon the webXR standard, a user-interaction programming interface that is shared across device vendors. This enabled us to support *hands*, *motion controllers*, and XR devices from as low as \$300 to \$4000. The third web standard is WebGL. We wrote a performant rendering, interaction, and programming interface from scratch using react-three/fiber, a modern lightweight and performant 3D framework with strong community backing. This has enabled real-time visualization of complex scenes and deformable objects. This technological landscape is fast-evolving — in the near future, when **webGPU** becomes fully supported, we will be able to parallelize the physics simulation using compute shaders with hardware acceleration. This will enable more complex physics that involve a large quantity of interacting particles, skin vertices, or liquid.

## 2.1 Multi-Physics Simulation in Vuer

At the core of Lucid-XR is vuer, a performant and simulator-agnostic XR framework running in the browser. We selected MuJoCo [2], a popular open-source physics simulation engine, and implemented a complete rendering front-end in react-three/fiber.

**Flexible Material and skin.** Existing data collection setup in virtual reality runs the physics simulation off-device in a separate computer [3]. This setup is workable if the number of poses involved is relatively small; however, when the scene includes flexible materials or a large number of particles, latency will increase, as more time is required to send the updated object data over the wire. Moving physics onto the device allows us to simulate flexible objects (see Figure 3).

**Simulating air.** MuJoCo has the ability to simulate rigid and deformable objects interacting with air. One can define a time-dependent wind vector. Figure 3c shows Lucid-XR running an example scene where fifty-two poker cards fall against a stationary capsule with the effect of air resistance in full display.

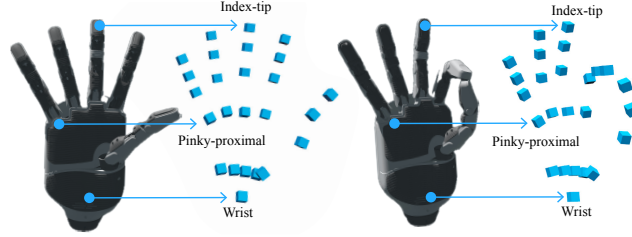


Figure 5: **Controlling Hand Pose via Motion Caption Sites.** We specify mocap sites by first aligning the proximal joints, and scale the hand so that fingers are similar in size as the robot hand. We then weld the  $SE(3)$  pose of the fingertip to similar sites on the robot hand and the wrist. We adjust the torque scale to balance position tracking and tracking the rotation of the pose.

83 **Collision modeling without convex decomposition.** Having to decompose a non-convex shape into  
 84 convex parts adds a step to the build. Many everyday objects, such as stairs, bowls, and mugs with  
 85 handles, tend to lose detail during convex decomposition. We include the signed-distance function  
 86 (SDF) collision solver plugin in MuJoCo by default, so that the user can always use SDF-based  
 87 collision without the convex decomposition step.

## 88 2.2 Precise Interactions at A Distance: Hitchhiking Controllers

89 The virtual world presents an opportunity to define new ways of interacting with robots. For in-  
 90 stance, the reference world frame changes at the beginning of each immersive session, which means  
 91 the gripper that the user wants to control is usually at a distance away from the user’s hands. Naïve  
 92 ways to control those grippers by “grasping” them from afar yields poor user experience, because as  
 93 the distance increases, the hand tracking error gets amplified. It is also difficult to apply rotation. We  
 94 solve this problem by applying  $SE(3)$  transformation to the target gripper in the local frame of the  
 95 mocap site. This design took inspiration from the *hitchhiking hand* [4], and let the activate mocap  
 96 sites at a distance by looking at the object and clicking on it. Once activated, the user can engage  
 97 with the site via a natural grasping gesture, where they close the lower three fingers.

## 98 2.3 On-Device Retargetting for Dexterous Hand Control

99 The problem of retargeting kinematic poses between humans and robots across different body kine-  
 100 matics is a common problem in robot data collection, which also appears in adjacent areas, including  
 101 legged locomotion and humanoid whole-body control [5]. Existing solutions rely on running the  
 102 kinematics solver separately on a server [6], making it hard to scale. Our solution involves binding  
 103 mocap sites to the tip of each finger and utilizing the relative pose to the wrist joint, as well as the  
 104 action space.

105 This hand control scheme is very general. We found that it worked well for all of the robot hands  
 106 that we experimented with. We designed a schema-based programming interface for users to specify  
 107 custom bindings between mocap sites or geometric bodies with landmarks and gestures of the hand  
 108 in python.

## 109 3 Synthesizing Diverse Manipulation Data from Virtual Demonstrations

110 The third component of Lucid-XR is a generative engine that converts the virtual human demonstra-  
 111 tions collected in the sim into diverse and realistic-looking multiview image datasets for the robot.  
 112 We follow the LucidSim [7] recipe that starts with a collection of diverse text prompts collected  
 113 from chatGPT, and use the semantic mask labels from the physics simulation to control the image  
 114 generation process.

115 **Generating realistic images from virtual demonstrations.** Figure 1 shows our setup for generat-  
 116 ing realist-looking images. In the insertion task, we create semantic masks paired with text prompts  
 117 describing the material and appearance of the object. These are sourced en masse from ChatGPT

118 via a meta-prompt (see appendix). Prompts for the background tend to be more complex. In align-  
 119 ment with observations made by prior works [7], we found it is key to generate these images from a  
 diverse set of text prompts.

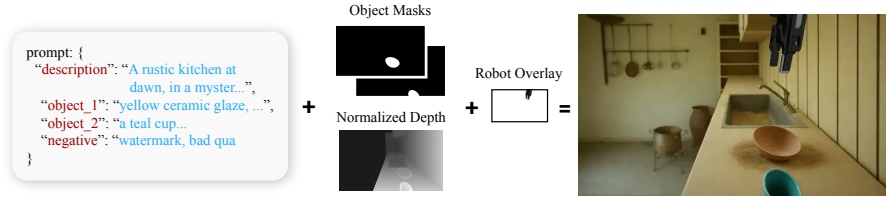


Figure 6: **Image generation pipeline.** We apply semantic masking and depth conditioning to precisely control the scene.

120

## 121 4 Results

122 Our analysis focuses on answering three key questions: First, what type of physics and robotic  
 123 form factors can Lucid-XR accomplish? Second, can we collect meaningful data to train policies to  
 124 perform tasks in sim? Third, can policies trained on Lucid-XR’s data work in realistic environments?

125 **Tasks** We created five contact-rich environments to evaluate on-device physics simulation in  
 126 Lucid-XR. Each environment tests a distinct type of physical interaction (see Fig. 7):

- 127 • **Pouring:** from a cup to a bowl – involve granular particles flow against rigid objects,
- 128 • **Fishing Toy:** Deformation of a flexible string into a knot with self-contact,
- 129 • **Tight Insertion:** millimeter-scale clearances in various peg-hole geometry,
- 130 • **Mug Rack:** modeling collisions with concave shapes using the SDF plugin,
- 131 • **Ball Sorting:** mixed rigid–particle collision in a toy sorter.

132 The vuer client maintains real-time simulation and rendering speed at 90 frames-per-second (fps) on  
 133 an Apple Vision Pro, thanks to the speed of WebAssembly, and the custom rendering frontend. Our  
 134 data is collected at 25 fps.

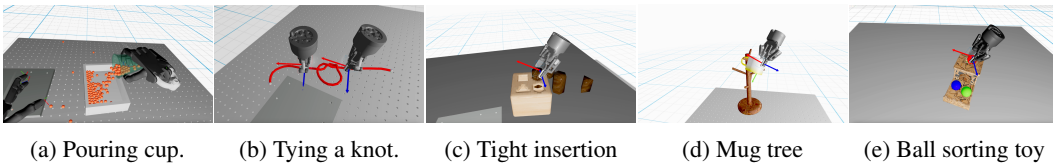


Figure 7: **Lucid-XR can simulate contact-rich manipulation of diverse physics.** (a) granular materials to simulate pouring liquid, (b) deformable materials when tying a knot, (c) tight tolerances between objects in contact, in a shape insertion task. (d) placing a mug onto a drying rack (e) a ball sorting toy.

### 135 4.1 Collecting Demonstrations and Training A Policy

136 We collected 30 minutes of demonstration data across six representative tasks: (a) *Pick-and-Place*  
 137 of a small cube, (b) *Block Stacking* (three-block tower), (c) *Place a mug in a drawer*, (d) *Valve*  
 138 *Turning*, (e) *Clearing a kitchen counter into the sink*, (f) *Ball Insertion* into a tight socket.

139 We are able to leverage the on-device kinematic retargeting to manipulate the two-finger grippers  
 140 and fully dexterous robot hands without any manual re-wiring or hardware swaps. Each motion  
 141 capture site was bound once in the scene schema, after which an operator could instantly switch

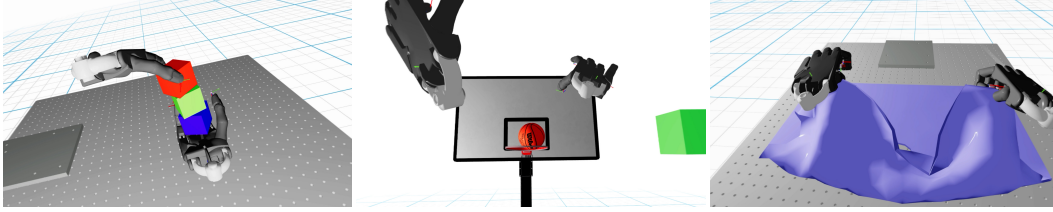


Figure 8: **Handling dynamic tasks and deformable objects.** The on-device retargetting is accurate enough to balance three blocks on one hand; is fast enough to handle dynamic tasks such as throwing a basketball, and handle deformable objects for cloth folding.

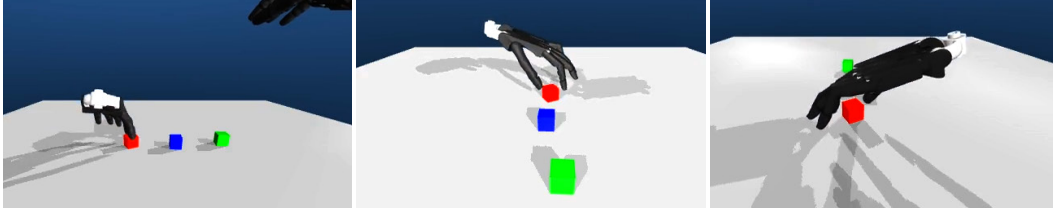


Figure 9: **Autonomous unroll from a learned dexterous manipulation.** We are able to train a dexterous manipulation policy to stack three colored blocks onto each other. We collect the relative pose of each fingertip in the reference frame of the wrist joint, and the global pose of the wrist landmark.

142 between end-effectors via the XR interface. In a real-world setup, resetting the robot and the scene  
 143 requires manually replacing the object, initializing the gripper pose, and safety checks. In vuer the  
 144 entire physics simulation and reset logic runs on-device, so the user can reset the environment by  
 145 pressing a “reset” button. This enables uninterrupted data capture for the full thirty minutes. Our  
 146 comparison shows a yield that is an order-of-magnitude more trajectories per task than would be  
 147 feasible on real hardware.

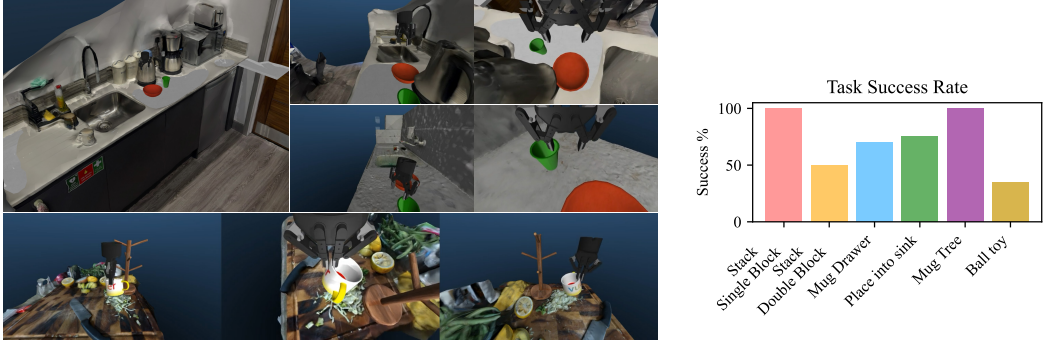
148 Using the standard ACT architecture [8], we trained one behavior-cloning policy per task. Each  
 149 ACT model receives as input the recorded proprioceptive state and camera views and outputs abso-  
 150 lute position commands. We optimized on expert ab, training each policy for 20k gradient updates  
 151 at a 25Hz control rate. The observation and actions spaces are the  $SE(3)$  pose of the gripper mid-  
 152 point. As our collected demonstrations are embodiment-agnostic. With this abstraction, the behavior  
 153 cloned policies can be deployed on any two fingered gripper and combined with data from any em-  
 154 bodiment.

## 155 4.2 Real-to-Sim Evaluation

156 We built a small set of intentionally cluttered and poorly lit evaluation environments to provide a  
 157 rough estimate of how well our policy generalizes. We made use of publicly available Polycam scans  
 158 from the internet, ensuring these meshes were noisy, realistic, and never-before-seen. We overlaid  
 159 these meshes on the original environment changing the visual distribution drastically. These envi-  
 160 ronments are intentionally cluttered and come with bad lighting. The policy trained on lucid-XR’s  
 161 data achieves success on never-before-seen visual domains with the use of the image generation  
 162 pipeline, while the raw ACT policy does not.

## 163 5 Related Works

164 This work took inspiration from several prior works. Most notably, our hitchhiking controller that  
 165 enabled users to teleoperate robot grippers from a distance builds upon the “hitchhiking hands” [4],  
 166 where users can interact with virtual hands by gaze. Our prompt and image generation pipeline is  
 167 adapted from LucidSim [7], with an expanded number of semantic classes and randomized object



(a) **Real-to-sim evaluation.** We used 3D scans of messy, real-world environments and placed them as visual backgrounds into the simulation to provide a rough estimate of the generalization capability of the trained policy.

(b) Success rate for each manipulation task, as measured in our extended-reality data engine.

Figure 10: Evaluation results from real-to-sim transfer (left) and per-task performance in simulation (right).

Kitchen Clearing	Base Env.	Low Clutter	High Clutter + Noise
ACT Policy	100%	0%	0%
ACT + LucidSim	100%	90%	25%

Table 1: Evaluation scores for the kitchen clearing task with unseen real-life meshes overlaid on the original environment. We score the Kitchen Clearing task based on two pick-and-places, allowing for 4 points per run.

168 geometry sampled from a database of 3D assets. Most closely related to our work is the DART  
 169 system [9], which, like our system, is a toolkit for collecting robot demos in virtual reality. A  
 170 primary difference between DART and our setup is that we run the simulation on-device, rather  
 171 than on the cloud. Simulation on the cloud incurs a latency between the actions the user takes and  
 172 simulated responses. This latency makes dexterous and dynamic control more difficult in practice on  
 173 the DART system along with large and collision-heavy scenes. We envision LucidXR as a system for  
 174 scaling up data collection for imitation learning using the power of generative models. Works such  
 175 as [8] provide a foundation for verified architectures and methodology for imitation learning tasks,  
 176 such as the Action Chunking Transformer[8]. The idea of using generative models to augment data  
 177 generation has also been explored by works such as [10, 11, 12, 13, 14] that make use of language  
 178 generation, image generation, asset generation, etc. to drive training.

## 179 6 Conclusion

180 In this work, we present Lucid-XR, a generative-AI-powered learning pipeline for producing gen-  
 181 eralizable visual policies for manipulation. We demonstrate the potential for virtual demonstrations  
 182 in a simulated world to produce real-world robot policies that generalize across object instances,  
 183 appearance, and lighting conditions. We also present new designs for interacting with virtual robots  
 184 through VR controllers and hand gestures. We believe virtual demonstration data has the potential  
 185 to scale across the internet and close the data gap for training a generally capable robot foundation  
 186 model.

## 187 7 Limitations

188 This work leaves a few rocks untouched. For instance, we rely on the controlling power of the same  
 189 text prompt to generate consistent views. A key benefit of generating visual data is that it comes  
 190 with paired text labels. Future iterations can benefit from the additional supervision that this pairing  
 191 provides.



## References

- [1] A. R. Cannon and B. Zachernuk. Introducing natural input for webxr in apple vision pro, Mar. 2024. URL <https://webkit.org/blog/15162/introducing-natural-input-for-webxr-in-apple-vision-pro/>. Accessed: 2025-05-01.
- [2] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi:10.1109/IROS.2012.6386109.
- [3] Y. Park, J. S. Bhatia, L. L. Ankile, and P. Agrawal. Dexhub and dart: Towards internet scale robot data collection. *ArXiv*, abs/2411.02214, 2024. URL <https://api.semanticscholar.org/CorpusID:273821640>.
- [4] R. Ban, K. Matsumoto, and T. Narumi. Hitchhiking hands: Remote interaction by switching multiple hand avatars with gaze. In *SIGGRAPH Asia 2023 Emerging Technologies*, pages 1–2. 2023.
- [5] X. Cheng, Y. Ji, J. Chen, R. Yang, G. Yang, and X. Wang. Expressive whole-body control for humanoid robots, 2024. URL <https://arxiv.org/abs/2402.16796>.
- [6] X. Cheng, J. Li, S. Yang, G. Yang, and X. Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024.
- [7] A. Yu, G. Yang, R. Choi, Y. Ravan, J. Leonard, and P. Isola. Learning visual parkour from generated images. In *8th Annual Conference on Robot Learning*, 2024.
- [8] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.
- [9] Y. Park, J. S. Bhatia, L. Ankile, and P. Agrawal. Dexhub and dart: Towards internet scale robot data collection, 2024. URL <https://arxiv.org/abs/2411.02214>.
- [10] P. Katara, Z. Xian, and K. Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023. URL <https://arxiv.org/abs/2310.18308>.
- [11] Y. Wang, Z. Xian, F. Chen, T.-H. Wang, Y. Wang, K. Fragkiadaki, Z. Erickson, D. Held, and C. Gan. Robogen: Towards unleashing infinite data for automated robot learning via generative simulation, 2024. URL <https://arxiv.org/abs/2311.01455>.
- [12] Y. J. Ma, W. Liang, H.-J. Wang, S. Wang, Y. Zhu, L. Fan, O. Bastani, and D. Jayaraman. Dreureka: Language model guided sim-to-real transfer, 2024. URL <https://arxiv.org/abs/2406.01967>.
- [13] Z. Chen, A. Walsman, M. Memmel, K. Mo, A. Fang, K. Vemuri, A. Wu, D. Fox, and A. Gupta. Urdformer: A pipeline for constructing articulated simulation environments from real-world images. *arXiv preprint arXiv:2405.11656*, 2024.
- [14] T. Yu, T. Xiao, A. Stone, J. Tompson, A. Brohan, S. Wang, J. Singh, C. Tan, D. M. J. Peralta, B. Ichter, K. Hausman, and F. Xia. Scaling robot learning with semantically imagined experience, 2023. URL <https://arxiv.org/abs/2302.11550>.

## 230 Appendix

231 We include additional simulated environments; details on how we engineer the physics involved; in  
232 addition to details on the generative workflow in this appendix. We also include examples of the  
233 virtual demonstrations that we collected, and examples of the synthetic images used for training. A  
234 live version of the presentation is included in The accompanying video.

### 235 7.1 Examples of Dexterous Virtual Demonstrations

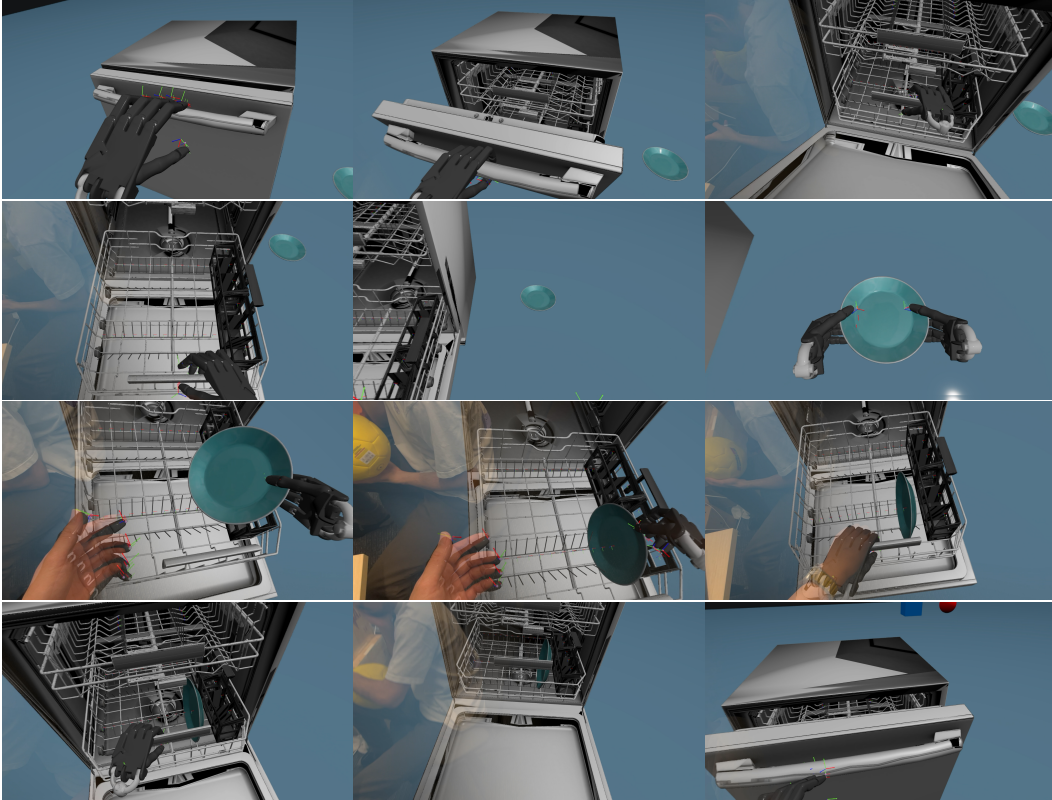


Figure 11: **Loading a Dishwasher.** Showing dexterous manipulation of a dishwasher with articulated doors a plate, and its interaction with the racks inside the dishwasher.

### 236 7.2 Examples of Policy Unroll in Simulation

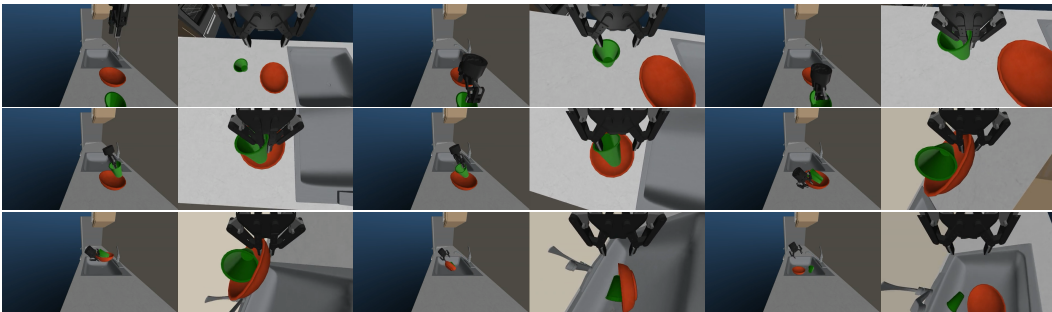


Figure 12: **Cleaning The Kitchen.** The robot learns to pick up a cup and stack it on top of the bowl, followed by placing both objects into the kitchen sink.

### 237 7.3 Examples of Learned Re-try Behavior

238 We noticed robust, re-try behavior from the learned policy. We present the image sequence in Figure. 13.

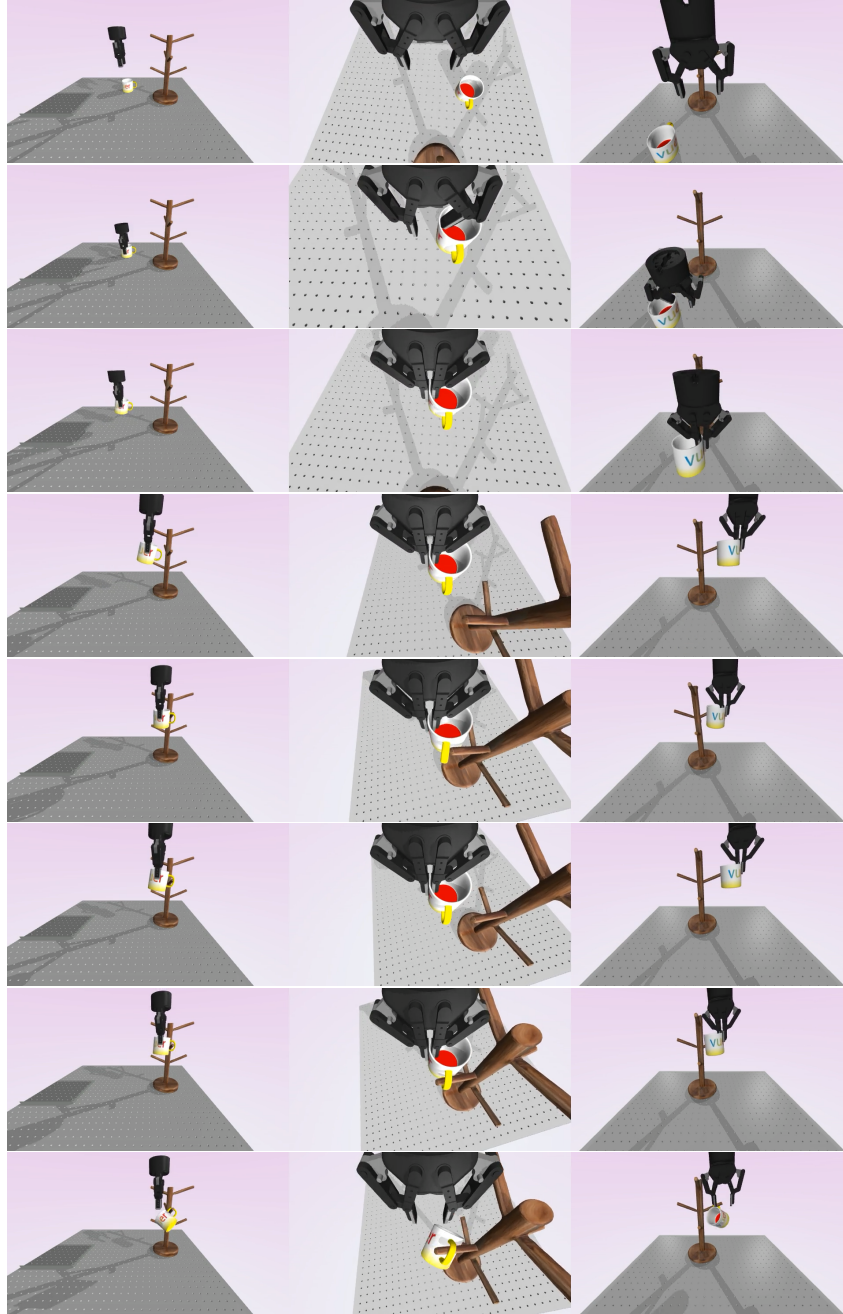


Figure 13: **Re-try Behavior in the Mug Tree Environment.** The policy came into contact twice: First during the picking up of the mug; the second time is when trying to hang the mug onto the arm of the mug tree drying rack.

## 7.4 Additional Examples of Synthetic Data

We include additional examples of synthetic images generated by the LucidXR pipeline in Figure 14.



Figure 14: **Additional Examples of Generated Images from Lucid-XR.** Notice the control over lighting, geometry, and content diversity.

## 7.5 The Vuer Scene Description Language

Usage of the MuJoCo engine API in Python tends to follow an imperative pattern, where objects, material texture, and lighting are changed by mutating MuJoCo’s physics and modeling buffers. To improve the readability and reusability of the MuJoCo simulation, we developed a declarative scene description language that treats the scene as a nested set of scene components that form a tree. This module, `vuer-mujoco`, enables the user to sketch out a scene via the following:

```
249 from vuer_mujoco import Box, DefaultScene, SimpleTable
250 1
251 2 from your_code import make_camera_rig
252 3
253 4 camera_rig = make_camera_rig()
254 5 lighting_rig = make_lighting_rig()
255 6
256 7 table = SimpleTable(pos=[0.0, 0.0, 0.0])
257 8
258 9 # a 10cm cube, initialized slightly above a table.
259 10 cube = Box(size=[0.01, 0.01, 0.01], pos=[0., 0.1, 0.02] + table.
260     surface_origin)
261 11
262 12 scene = DefaultScene(
263 13     *camera_rig.get_all_cameras(),
264 14     lighting_rig.key, lighting_rig.fill, lighting_rig.back,
265 15     table, cube, robot, robot.mocap_points,
266 16 )
267 17
```



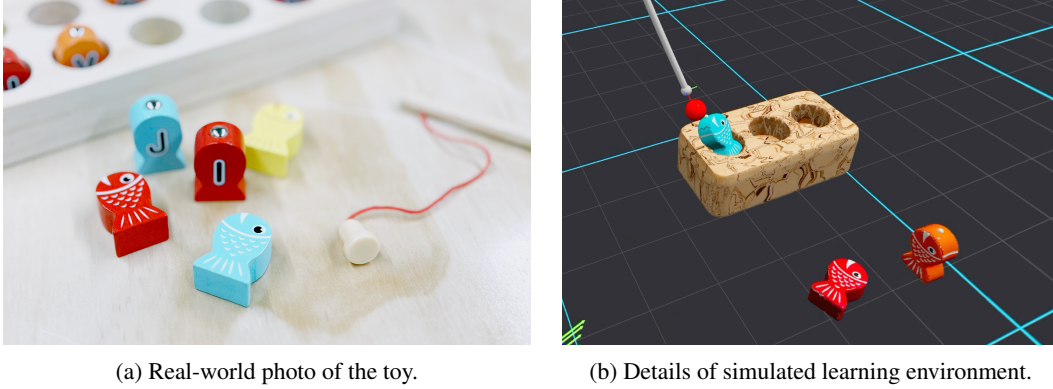


Figure 15: The real-world fishing toy set (left), versus the simulated learning environment used to collect data (right). The fishing rod has a magnetic hook attached to the end of the line that interacts with a ferromagnetic nail head embedded in the fish. We simulate this by defining adhesion actuators on a small sphere body attached to the fish geometry.

## 7.6 Tips on Engineering Synthetic Environments

**Adhesion.** Adhesive interactions are involved in many physical processes, including surface tension, magnetic interaction with ferro- and para-magnetic materials, interaction with sticky surfaces, such as an object against the silicone rubber material in a gripper.

We provide a fishing toy environment that simulates a toy fishing pole with a magnetic hook that the robot can use to pick up a wooden fish. This simultaneously involves interaction with flexible material – a rope. We set the weight of the fish pieces so that the adhesion is strong enough to lift them off the table, but still weak enough to detach with a slight shake of the fishing rod.

**Generating 3D Assets.** All of the 3D assets shown above are generated either from a text description of the object or stock images of the physical item found on Amazon.com. We use a free version of the meshy.ai service. The resulting 3D meshes usually contain a large number of faces, with physical dimensions that are off by two orders of magnitude. We post-process these 3D assets by first simplifying them in MeshLab, and then centering and rescaling using a custom script.

```
from vuer_mujoco import MuJoCoRope

rope = MuJoCoRope(
    postamble="""
    <actuator>
      <adhesion body="f1-magnet" ctrlrange="0.15 0.16" gain="1"/>
      <adhesion body="f2-magnet" .../>
      <adhesion body="f3-magnet" .../>
    </actuator>
    """,
)
```

## 7.7 Example Image Prompts

We produce a small number of randomly selected text prompts below.

**Prompts for the kitchen cup/bowl Scene:**

```
bowl = "A rich green patinated copper bowl, its surface embellished
with intricate embossing echoing the craftsmanship of generations
past, each detail wrought with precision."

cup = "A minimalist acrylic cup, hard-edged and visually weightless,
mimicking the ethos of contemporary transparency."
```



```

304 4
305 5 background = "A chaotic ensemble of baking supplies lie scattered
306     around, flour dusts the edges of a marble countertop, while
307     reflective surfaces from an overhead light palette interact
308     sporadically creating patterns on the brushed steel faucet."

```

Listing 1: **The Crossroad of Time.** Illustrating the intersection of aesthetic past and present through culinary objects. (Minimalist cup, antique bowl)

```

310
311 1 bowl = "A ceramic bowl with swirling patterns of cobalt blue,
312     sunflower yellow, and rose pink glazes, with a high-gloss finish."
313 2
314 3 cup = "A teacup marked by abstract patches of emerald green and
315     crimson red, seemingly creating a kaleidoscopic effect."
316 4
317 5 background = "A backdrop filled with chaotic elements \u2014 a
318     countertop cluttered with spatulas, over-ripe fruit, and colorful
319     cracked ceramic tiles leaning against the wall."
320

```

Listing 2: **Mosaic Glaze Fantasy.** A close-up of vibrant ceramics with a dazzling faucet. (Colorful ceramics, shining faucet)

**Prompts for the mug tree environment:**

```

321
322
323 1 mug = "An oversized mug with a faded \u2018World's Best Bartender\
324     \u2019 logo, filled with room-temperature coffee and a tiny
325     lipstick mark on its rim."
326 2 mug_tree = "A sleek, modern steel drying rack holding a neat row of
327     eclectic mugs, each hanging at a slightly different angle."
328 3 background = "The counter is scattered with an array of bar staples,
329     including an upturned shaker, a jar of maraschino cherries left
330     open with sticky syrup pooled at the base, tiny, colorful drink
331     umbrellas laying flat, and cocktail recipe cards partially
332     obscured by a dishcloth. The dim overhead lighting casts a cozy
333     yet neglected ambiance, reflecting off of glass surfaces,
334     highlighting water stains and the subtle sheen of unpolished wood.
335     Seasonal drink posters curl at the corners on the walls behind,
336     and the muted hum of conversation drifts from unseen patrons."
337

```

Listing 3: **Afternoon Clutter.** An up-close view of a neglected bar counter amidst a busy afternoon, capturing the hustle and bustle pausing for just a moment. (bar counter)

```

338
339 1 mug = "A tall novelty mug with light visible scuffs, its personality
340     yet vibrant, with a traveler's emblem boldly emblazoned.",
341 2 mug_tree = "Innovatively designed transparent drying rack, its
342     architecture raising daily-use mugs to a dignified height",
343 3 background = "The bar counter's imperfections speak of immediate use \
344     \u2013 glistening droplets from pint glasses left haphazardly,
345     several salt grains shining under muted sunlight, and a stray beer
346     tab nestled amongst dried orange zest. A sharp, storied scratch
347     veers perilously close to a vase of gaudy red carnations.
348     Decorative but decisively aged beer mats pattern the visible
349     stained wood, while a distant chorus of cheerful banter tingles
350     the air.",
351

```

Listing 4: **Unexpected Afternoon Clutter.** A surprisingly busy scene of a bar counter overtaken by mid-day merriment. (bar counter)

## 7.8 Image Generation Workflow

We provide the complete image generation workflow in JSON form in the supplementary material. This workflow can be loaded into ComfyUI as-is. A screenshot fo this workflow can be found in Figure. 16

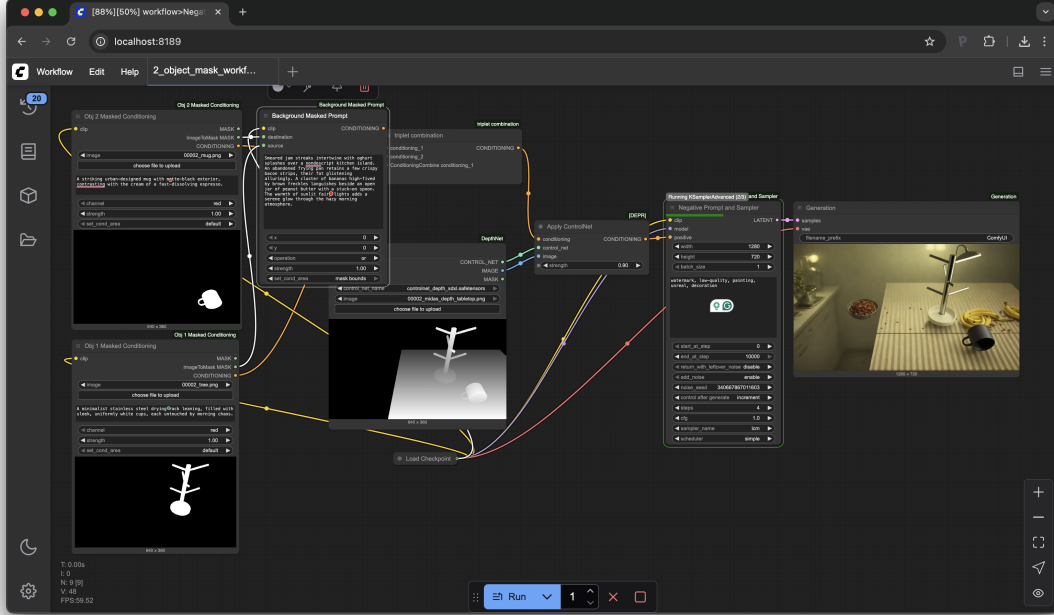


Figure 16: **Image Generation Workflow.** Using two object masks plus a normalized inverse depth image, we are able to control the geometry, lighting, and the composition of the generated images.

## 7.9 Real-to-sim Evaluation Setup

We include two realistic testing environments: a) Clean Kitchen and b) Messy Kitchen. These environments serve as a more controllable proxy of the real-world experiments, to enable faster iteration on the data and learning pipeline.

We align the 3D mesh from PolyCam to the MuJoCo environment (see Figure. 18) manually, shown in Figure 17.

## 7.10 Training Details

We use the Action-Chunking transformer architecture from [8]. Our policy uses multiple backbones (for each input camera feed). The hyperparameters we used are in Table. 2.

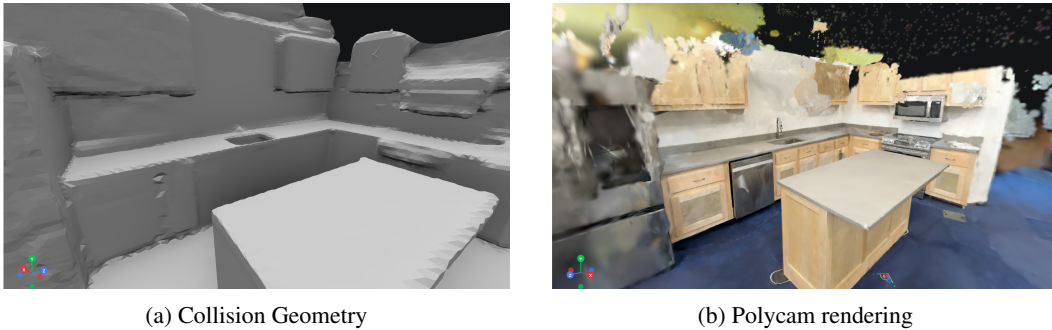
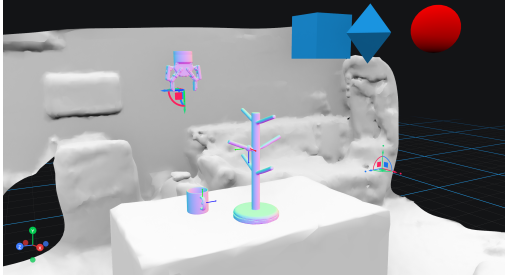
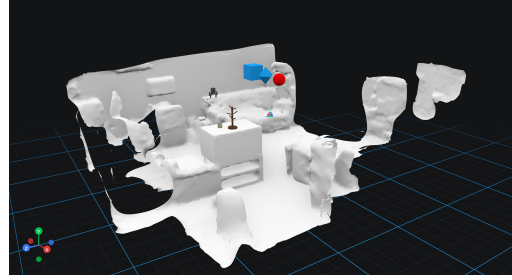


Figure 17: **Clean Kitchen.** Contain no clutter. We place 3D object assets programmatically.



(a) Aligning the 3D mesh with the MuJoCo scene.



(b) The scene after alignment.

Figure 18: **Aligning scan with the physics environment.** .

Table 2: Training Hyperparameters.

<b>learning rate</b>	5e−5
<b>batch size</b>	32
<b>number of encoder layers</b>	4
<b>number of decoder layers</b>	7
<b>feedforward dimension</b>	3200
<b>hidden dimension</b>	512
<b>number of heads</b>	8
<b>chunk size</b>	10
<b>KL-weight</b>	10
<b>dropout</b>	0.1