SAGE: Scalable Agentic 3D Scene Generation for Embodied AI

Supplementary Material

Appendix

A. Additional Experiment Results

In this section, we present additional visualization results and implementation details for both scene generation (Sec. A.1, A.2, A.3) and robot learning (Sec. A.4), followed by runtime analysis (Sec. A.5). We encourage readers to visit our project webpage for additional visualization videos.

A.1. SAGE-10k Dataset

To support community research at scale, we pre-generated a 10k-scene dataset with our approach, named SAGE-10k Dataset. It's across 50 room types in diverse styles, including 565K uniquely generated 3D objects. The preview image and statistics of the dataset are shown in Fig. 1.

A.2. Additional Capabilities and Extensions

In this subsection, we present several straightforward extensions of SAGE that demonstrate its flexibility across diverse generation settings. While these capabilities are not the primary focus of this paper, they arise naturally from the modular design of our framework and require minimal adaptation. These results highlight the generality of SAGE and suggest that further improvements in generation quality can be achieved by strengthening individual components, which we leave as a promising direction for future work.

Multi-room Layout We showcase the multi-room generation capability of SAGE across different kinds of scene layouts, as Fig. 2 shows. Connected floor plans are created in the Scene Initializer. Additionally, the agent can execute parallel updates with Asset Placer/Mover/Remover, and the generators are updated to accept multiple room IDs and conditions within a single MCP tool call.

Image-conditioned Scene Generation We demonstrate that SAGE can be extended with image-conditioned generation without architectural changes by utilizing Qwen3-VL [10] to extract style and object attributes from reference images, as shown in Fig. 3. While not pixel-aligned, the generated scenes remain semantically coherent.

Articulated Objects In addition to rigid-body objects, our modular design can easily extend the current text-to-3D generation with object retrieval. As Fig. 4 shows, SAGE can be integrated with articulated assets from PartNet-Mobility [12] in the generated scenes. Also, robot actions can be generated by grasp pose prediction and motion planning.

A.3. Scene Augmentation

In this section, we will show more results related to our proposed scene augmentation method, including Object Category-level Augmentation and Scene Layout-level Augmentation.

A.3.1. Object Category-level Augmentation

For a previously generated single base scene produced by our agentic scene generation framework, we can augment the task relevant objects with different shapes and textures while keeping the same object category, so that robot can learn generalizable policy to unseen scenarios. Here we showcase the capability of our category augmentation method. To illustrate the capacity of our augmentation method, we randomly select part of the objects in the scene for category augmentation. Following the method we described in the main paper method section, given the text description of the selected object from the generation stage, we employ an LLM-based text augmentation to produce variations in geometry and texture (*e.g.*, shape, color, material, or finish) while maintaining the original object category. We then use TRELLIS [13] to synthesize corresponding 3D assets from these augmented descriptions, which are placed into the scene to enrich visual and physical diversity across instances. We show the object category-level augmentation results in different scene categories in Fig. 5.

A.3.2. Scene Layout-level Augmentation

In addition to object category-level augmentation, to diversify the background environment as well besides the task-relevant objects, we proposed scene layout-level augmentation. The background scene, including room geometry and all task-irrelevant objects, is regenerated through the agent-driven scene generation, while we keep the task-relevant objects unchanged but repositioned according to the new scene layouts. This process produces diverse scene layouts sharing the same task specification, enabling learning policies that generalize across spatial configurations. We show the scene layout-level augmentation results in different scene categories in Fig. 6.

A.4. Robot Action Generation and Policy Inference

A.4.1. Robot Action Generation

We also showcase our parallel robot action generation process with the scaled training data in our project webpage. The collected robot action data is composed of robot end-effector poses with multiple camera views. For Pick-and-Place task, the camera views input includes 3 perspective cameras located at left, right, and wrist of the Franka Emika Panda

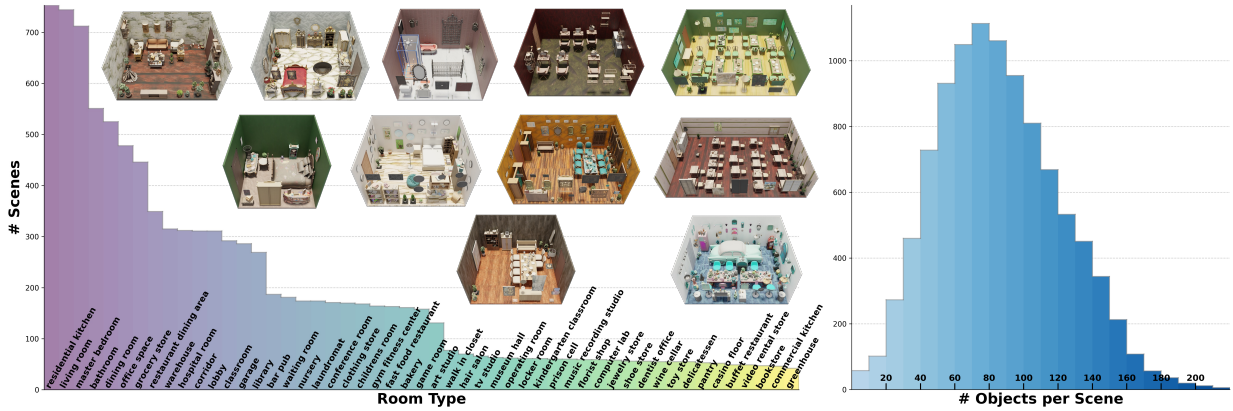

Figure 1. **SAGE-10k Dataset**: We pre-generated a 10k-scene dataset named SAGE-10k Dataset across 50 room types and 50 styles, including 565K uniquely generated 3D objects. We include the statistics of room types, room examples, and objects per scene in the figure as well. The dataset can be accessed via this link.

"Multilingual teacher's apartment" "A student apartment with one bedroom" "Mid-century modern family home"

Figure 2. **Multi-room open-vocabulary generation.** SAGE can be extended to generate multi-room scenes at scale easily by generating the floor plan and then calling generator MCP tools to fill in multiple rooms in parallel.

Figure 3. **Image-conditioned scene generation.** Using Qwen3-VL [10], SAGE extracts style and object attributes from reference images to enable image-conditioned scene generation without architectural modifications. The generated scenes are not pixel-aligned but remain semantically consistent with the reference images.

robot. For Mobile Manipulation task, the camera views input includes 5 cameras. Three of them are perspective cameras located at left, right, and wrist of the composed mobile manipulator robot. The extra two cameras are fisheye cameras located on top of the robot, looking front and back separately for navigation purposes. We input both RGB and depth from each view with the resolution of 128×128 to the policy network [1]. Parallelized action generation is performed with 8 environments for Pick-and-Place task and 2 environments for Mobile Manipulation task in parallel per GPU in the IsaacSim [8] simulator.

A.4.2. Policy Inference

For policy inference, we feed the trained policy network with the observed camera views and execute the inferred robot actions. We showcase the policy inference demos in our project webpage, for both the Pick-and-Place task and the Mobile Manipulation task. Failure cases are mostly due to the randomness of the policy inference, far-away objects, or difficult grasp pose location which is hard to reach and grasp by the robot.

A.5. Runtime Analysis

Scene Generation Generation time for a single scene with SAGE varies according to user demands. The most time-

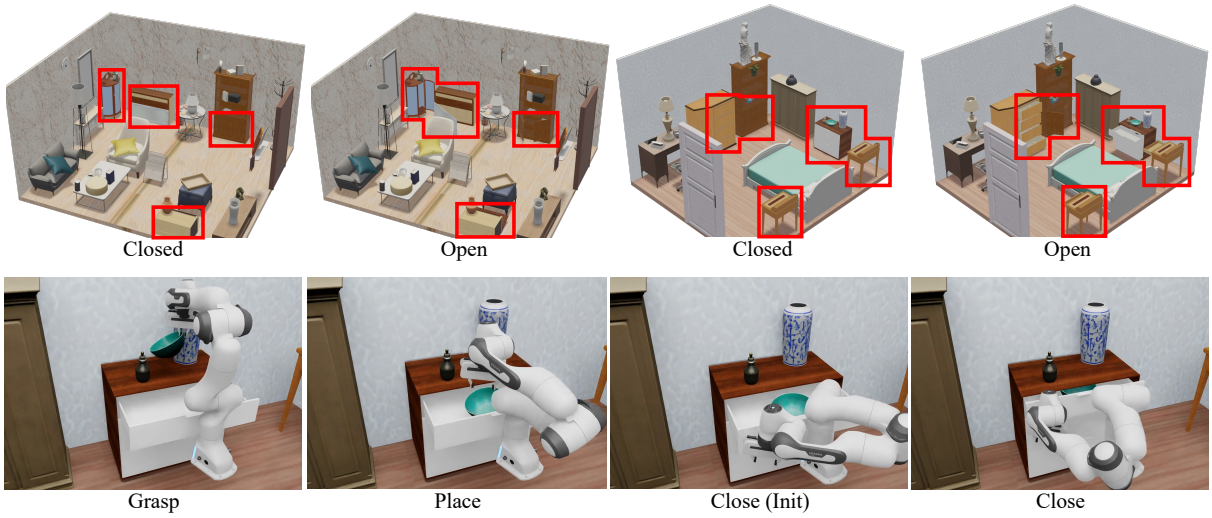

Figure 4. **Articulated Objects**: SAGE can be extended with articulated objects using retrieval from PartNet-Mobility [12]. *Top*: we show two scenes with multiple articulated objects at closed and open states. *Bottom*: an action sequence generated with grasp pose prediction and motion planning(Sec.**??**) for "pick up the bowl, place it in the drawer, and close the drawer". Please visit website for full video.

consuming components are object generation and simulation-in-the-loop validation for physical stability, while agent reasoning and LLM/VLM inference are faster. Object generation with TRELLIS [13] takes 15 seconds per object. We parallelize this across 8 GPUs, reducing average generation time to 2-3 seconds per object. Simulation with Isaac Sim [8] takes 1-2 seconds per placement candidate for stability validation. To minimize simulation overhead, we simulate once after placing all floor and wall objects, then remove any unstable objects. This significantly improves efficiency by consolidating multiple simulations into one. For on-top objects, however, we simulate each placement individually since these objects are typically smaller and more prone to instability. We adopt an early-stopping strategy that accepts the first stable, collision-free placement validated by Isaac Sim [8]. This greatly reduces computation time, as we typically find stable placements within a few trials despite having 30-50 candidate locations. Overall, while generation time spans a broad range, a scene with 20 objects takes approximately 10 minutes, with time scaling linearly for additional objects.

Action Generation Using Isaac Lab [6] as our simulation platform, we leverage its parallelism to significantly reduce data collection time. For Pick-and-Place tasks, motion planning for each demonstration takes 8-10 seconds per environment when run sequentially. By parallelizing across 8 environments, the total planning time increases to 15-20 seconds, but this yields an average of 2-3 seconds per demonstration. For Mobile Manipulation tasks, which involve longer trajectories, the parallelized per-demonstration time is 8-10 seconds. Our simulations scale readily across GPU clusters, enabling proportional speedup in data generation based on available GPU resources. This scalability facilitates efficient collection of large-scale datasets for training generalizable policies.

Policy Training We train a diffusion policy [1] using the Robomimic [5] framework, which takes several hours to converge on our robot action data. Note that diffusion policy is one approach to convert our generated actions into an executable policy, chosen here for its simplicity. Other methods, such as fine-tuning a VLA, may prove more efficient and we leave for future work.

B. Additional Implementation Details

In this section, we will elaborate the implementation details of our proposed agent-driven scene generation and robot action generation as well as policy learning.

B.1. Scene Generation

B.1.1. Overview

Our scene generation framework uses a carefully designed agent-driven system. The SAGE operates under the Model Context Protocol (MCP) [7], a standardized protocol for seamless interaction with external tools [7]. For the agent to understand each tool, we provide descriptions that include the tool's function, input argument types (specified as Python strings), and output format. We formulate all tool outputs as JSON dictionary strings, which the agent can easily parse and interpret. At each iteration, the agent either specifies

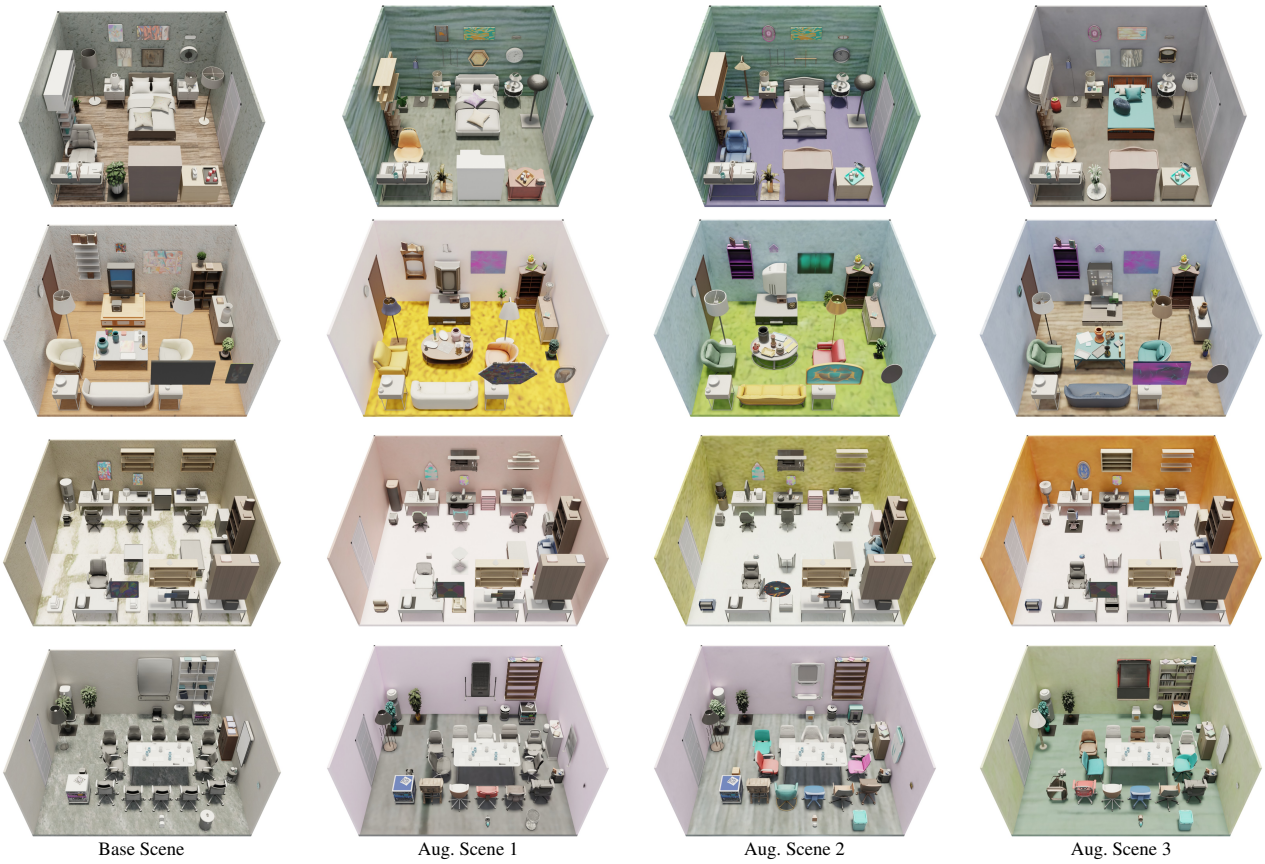

Figure 5. **Object Category-Level Augmentation.** Here we showcase the capability of our category augmentation method. We randomly select part of the objects in the scene for category augmentation. Given the text description of the selected object from the generation stage, we employ an LLM-based text augmentation to produce variations in geometry and texture (*e.g.*, shape, color, material, or finish) while maintaining the original object category. We then use TRELLIS [13] to synthesize corresponding 3D assets from these augmented descriptions, which are placed into the scene to enrich visual and physical diversity across instances.

the next tool and its input arguments, or indicates that scene generation is complete. The server executes the requested operation and returns the result, which the agent uses as feedback to determine the next action. This setup enables adaptive, tool-driven scene generation without hard-coded logic. For the sake of spaces in supplementary, we will include all the prompts we used in our code release.

B.1.2. Generators

The scene is constructed using a set of generator tools that the agent dynamically invokes via MCP. Each generator is an MCP tool the agent can call at each iteration. Below we describe the implementation of each generator.

Scene Initializer The scene initializer generates the floor plan and materials for the floor and walls. Floor plan generation proceeds in two steps. First, we prompt the LLM with the scene description, which returns the room types and sizes. For multi-room layouts, a second step generates connectivity between rooms and places connecting doors. We generate materials from text descriptions of the floor and walls provided by the LLM using Matfuse [11]. In the supplementary material, we also use Flux.1-dev [2, 3] to generate more detailed textures. We then assemble the floor plan with the generated materials to create the scene layout.

Asset Placer The asset placer takes a text string describing placement requirements for objects, generates them with TRELLIS [13], and places them into the 3D scene. After the text-to-3D inference with TRELLIS [13], we perform a series of mesh post-processing, including decimation, non-manifold correction, and hole filling, to ensure the watertight property of each generated mesh. After asset generation, we use a VLM to estimate physical properties, including height, mass, and metallic/roughness values. For each object, an LLM analyzes the input conditions and categorizes the placement as floor, wall, or on-top. The placement logic varies by category. For floor objects, we first use the LLM to generate constraints, including global position in the room and relative position/orientation to existing objects. These con-

Figure 6. **Scene Layout-Level Augmentation.** Here we show more results of Scene Layout-Level augmentation, where the background scene, including room geometry and all task-irrelevant objects, is regenerated through the agent-driven scene generation. This process produces diverse scene layouts sharing the same task specification, enabling learning policies that generalize across spatial configurations. **Bedroom:** We keep the objects of *desk*, *nightstand*, and *mug on the nightstand* as the same. **Livingroom:** We keep the objects of *sideboard*, *coffeetable*, and *vase on the coffeetable* as the same. **Office:** We keep the objects of *sofa*, *desk*, and *pen on the desk* as the same. **Meeting room:** We keep the objects of *table*, *cabinet*, and *cup on the meeting table* as the same.

straints guide our placement scoring. We sample available positions using a grid-based approach, then apply depth-first search to find optimal positions and orientations while checking for collisions. For wall objects, we similarly use depth-first search with grid-based sampling and collision checking against both floor and wall objects. For on-top placement, unlike [14] which supports only single-layer relationships, our method enables multi-layer scene graphs. We sample available locations by computing surface normals on the supporting object's mesh, selecting faces with normals close to the room's up axis. This allows placement on shelves and surfaces, not just object tops as in [14]. After collision checking, we obtain multiple candidate locations. Finally, we validate stability using Isaac Sim [8]. In the simulation, all the wall objects are set as static since they are attached on the wall. We simulate each candidate placement—if unstable (*e.g.*, a pillow standing on a bed), we record the post-simulation pose and re-simulate with this adjusted pose. If the second simulation is stable, we accept the placement; otherwise, we reject it. If all candidates fail, the physics critic will report the failure to the agent and suggest alternatives such as smaller objects or different support surfaces. Note that unlike [14], which supports only single-iteration placement, our agentic framework enables iterative, adaptive placements across multiple tool calls to fully realize the scene and satisfy user requirements.

Asset Mover The Asset Mover locates and moves objects specified in the input text string. It first uses the integrated LLM to parse the target object and its destination. The object is temporarily removed from the current room, then repositioned using the same placement logic applied to other objects. If placement fails due to insufficient space, the object is restored to its original location and the failure is reported to the agent. The agent can then choose alternative actions, such as moving the object elsewhere or removing other objects to free up space. Movement instructions typically come from the visual critic.

Asset Remover The Asset Remover deletes objects described in the input text, using LLM reasoning to locate them in the scene. It is typically called when the critic provides feedback to remove an object.

B.1.3. Critics

Visual Critic The visual critic evaluates the current scene configuration and suggests new objects to place or adjustments to existing placements. When proposing new objects, it considers several factors. First, it identifies natural object combinations—for example, placing chairs around tables or books on bookshelves—using example combinations we provide to guide its reasoning. Second, when the room appears sparse, it suggests background elements like floor plants or decorative objects to fill empty spaces. Third, it verifies whether task-relevant objects specified by the user have been placed, notifying the agent if any are missing. For adjustments to existing objects, the critic analyzes rendered images to determine which items should be moved or removed. This feedback guides the agent in selecting the most appropriate generator to invoke next.

Physics Critic The physics critic operates at every stage of each generator, explicitly validating scene stability after object operations—addition, movement, and removal. During physical simulation, wall-mounted objects are treated as static since they attach to walls, while all other objects remain dynamic and respond to collisions and gravity. When operations fail due to physics instability, the physics critic detects these failures and returns them to the agent as feedback.

B.2. Scene Augmentation

We apply two types of augmentation to our agent-generated scenes: object category-level and scene layout-level. Implementation details are provided below.

Object Category-Level Augmentation We adopt TRELLIS [13] to generate augmented objects given the augmented text descriptions of objects. The generated objects are then placed into the scene with the same supporting relationships as their originals, and physics validation is performed using Isaac Sim [8]. Wall and floor textures are also able to be augmented during this process.

Scene Layout-Level Augmentation In layout-level augmentation, the background scene—including room geometry and all task-irrelevant objects—is regenerated through agent-driven scene generation, while previously generated task-relevant objects are preserved and reused. To achieve this, we add a stage in the Asset Placer generator that excludes pre-generated objects from the generation list using LLM-based reasoning, preventing them from being regenerated. This ensures that task-relevant objects are preserved while all task-irrelevant objects and the room layout are regenerated, producing diverse spatial configurations that enable policies to generalize better. Note that scenes from layout-level augmentation can undergo further augmentation. For example, we can combine them with another round of category-level augmentation to create even greater diversity.

B.3. Robot Action Generation

Pick-and-Place We use M2T2 [15] to generate grasp pose candidates from rendered depth images. These are then transformed using the camera pose to obtain the actual 3D grasp pose in world coordinates for motion planning with inverse kinematics (IK). Curobo [9] is integrated into the motion planning and IK pipeline by incorporating mesh geometries into its collision checking, ensuring feasible and stable grasp execution. Once we have the grasp pose, we divide the grasp into multiple steps. First, we use IK to calculate the end-effector trajectory from the start pose to a position directly above the grasp pose. Next, the gripper lowers and closes to grasp the object. Finally, the gripper lifts to raise the object. For placement, we use IK with Curobo collision avoidance. We simplify placement to a drop, which could be improved with additional motion planning steps to gently place the object down.

Mobile Manipulation This task is composed of a few subtasks including object grasping, navigation, and placement. Object grasping follows the same motion planning as Pick-and-Place. For navigation, we use RRT [4] to plan collision-free trajectories. The algorithm explores possible paths from both start and target positions, ending when the paths meet. Collision checking is implemented using a 2D occupancy grid, which is faster and saves memory compared to 3D explicit mesh collision checking.

Parallelism We leverage parallelism features in Isaac Sim and Isaac Lab [6, 8] to simulate multiple environments in parallel per GPU (8 for Pick-and-Place and 2 for Mobile Manipulation). This scales easily to multiple GPUs in a cluster. We use NVIDIA L40S GPUs in our experiments.

References

[1] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion, 2024. 2, 3

[2] Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2024. 4

[3] Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, Sumith Kulal, Kyle Lacey, Yam Levi, Cheng Li, Dominik Lorenz, Jonas Müller,

Dustin Podell, Robin Rombach, Harry Saini, Axel Sauer, and Luke Smith. Flux.1 kontext: Flow matching for in-context image generation and editing in latent space, 2025. 4

[4] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998. 6

[5] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021. 3

[6] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit - A Unified Simulation Framework for Interactive Robot Learning Environments. *IEEE Robotics and Automation Letters*, 8 (6), 2023. 3, 6

[7] Model Context Protocol (MCP). Model Context Protocol: Open protocol that standardizes how applications provide context to llms. https://modelcontextprotocol.io/introduction, 2025. Accessed: 2025-11-11. 3

[8] NVIDIA Corporation. Nvidia isaac sim. https://github.com/isaac-sim/IsaacSim, 2025. Version 5.0.0. 2, 3, 5, 6

[9] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023. 6

[10] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 1, 2

[11] Giuseppe Vecchio, Renato Sortino, Simone Palazzo, and Concetto Spampinato. Matfuse: Controllable material generation with diffusion models. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4429–4438. IEEE, 2024. 4

[12] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020. 1, 3

[13] 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, 2025. 1, 3, 4, 6

[14] Yue Yang, Fan-Yun Sun, Luca Weihs, Eli VanderBilt, Alvaro Herrasti, Winson Han, Jiajun Wu, Nick Haber, Ranjay Krishna, Lingjie Liu, Chris Callison-Burch, Mark Yatskar, Aniruddha Kembhavi, and Christopher Clark. Holodeck: Language guided generation of 3d embodied ai environments, 2024. 5

[15] Wentao Yuan, Adithyavairavan Murali, Arsalan Mousavian, and Dieter Fox. M2t2: Multi-task masked transformer for object-centric pick and place, 2023. 6