
Scaling Embodied Artificial Intelligence

Massive 3D Simulations to Real-World Distributional Robustness

Matt Deitke

Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle
matt@cs.washington.edu

Bachelors Thesis
June 2023

Advisor: Ali Farhadi



Contents

1 ProcTHOR	5
1.1 Introduction	5
1.2 Related Work	6
1.3 PROCTHOR	7
1.4 PROCTHOR-10K	11
1.5 ArchitecTHOR	12
1.6 Experiments	13
1.7 Conclusion	13
2 Objaverse	15
2.1 Abstract	15
2.2 Introduction	15
2.3 Related Work	17
2.4 Objaverse	18
2.5 Applications	21
2.5.1 3D Generative Modeling	21
2.5.2 Instance Segmentation with CP3D	22
2.5.3 Open-Vocabulary ObjectNav	23
2.5.4 Analyzing Robustness	24
2.6 Conclusion	26
3 Phone2Proc	27
3.1 Abstract	27
3.2 Introduction	27
3.3 Related Work	29

CONTENTS	3
3.4 Approach.	30
3.4.1 Scanning.	30
3.4.2 Environment-Conditioned Procedural Generation	31
3.4.3 Transfer to Real World	32
3.5 Experiments.	34
3.5.1 How Well Does Phone2Proc Work?	34
3.5.2 How Does Phone2Proc Compare To A Privileged Upper Bound?	35
3.5.3 How Robust is Phone2Proc to Chaos?	35
3.5.4 Statistical Analysis	36
3.5.5 Qualitative Analysis	37
3.5.6 Quantitative Analysis of The Environments	38
3.6 Conclusion	38
A ProcTHOR Appendix	56
A.0.1 Room Specs.	62
A.0.2 Sampling Floor Plans	62
A.0.3 Connecting Rooms	64
A.0.4 Structure Materials	65
A.0.5 Ceiling Height	66
A.0.6 Lighting	66
A.0.7 Object Placement	67
A.0.8 Material and Color Randomization	76
A.0.9 Object States	76
A.0.10 Validator	77
A.0.11 Related Works	77
A.0.12 Limitations and Future Work	80
A.1 PROCTHOR Datasheet	81
A.2 ARCHITECTHOR.	85
A.2.1 Datasheet	87
A.2.2 Analysis	90
A.3 Input Modalities	91
A.4 Experiment details.	92
A.4.1 ObjectNav experiments	92

<i>CONTENTS</i>	4
A.4.2 ArmPointNav experiments	96
A.4.3 Rearrangement experiments	97
A.5 Performance Benchmark	97
A.6 Robustness	98
B Objaverse Appendix	100
B.1 Instance Segmentation with CP3D	100
B.2 Open-Vocabulary ObjectNav	100
B.3 Composition	103
B.4 Estimating Coverage	104
C Phone2Proc Appendix	105
C.1 Implementation Details	105
C.2 Failure Cases	106

Chapter 1

ProcTHOR

1.1 Introduction

Computer vision and natural language processing models have become increasingly powerful through the use of large-scale training data. Recent models such as CLIP [196], DALL-E [200], GPT-3 [16], and Flamingo [3] use massive amounts of task agnostic data to pre-train large neural architectures that perform remarkably well at downstream tasks, including in zero and few-shot settings. In comparison, the Embodied AI (E-AI) research community predominantly trains agents in simulators with far fewer scenes [198, 127, 50]. Due to the complexity of tasks and the need for long planning horizons, the best performing E-AI models continue to overfit on the limited training scenes and thus generalize poorly to unseen environments.

In recent years, E-AI simulators have become increasingly more powerful with support for physics, manipulators, object states, deformable objects, fluids, and real-sim counterparts [127, 214, 223, 77, 266], but scaling them up to tens of thousands of scenes has remained challenging. Existing E-AI environments are either designed manually [127, 77] or obtained via 3D scans of real structures [214, 198]. The former approach requires 3D artists to spend a significant amount of time designing 3D assets, arranging them in sensible configurations within large spaces, and carefully configuring the right textures and lighting in these environments. The latter involves moving specialized cameras through many real-world environments and then stitching the resulting images together to form 3D reconstructions of the scenes. These approaches are not scalable, and expanding existing scene repositories multiple orders of magnitude is not practical.

We present PROCTHOR, a framework built off of AI2-THOR [127], to procedurally generate fully-interactive, physics-enabled environments for E-AI research. Given a room specification (e.g., a house with 3 bedrooms, 3 baths, and 1 kitchen), PROCTHOR can produce a large and diverse set of floorplans that meet these requirements (Fig. 1.1). A large asset library of 108 object types and 1633 fully interactable instances is used to automatically populate each floorplan, ensuring that object placements are physically plausible, natural, and realistic. One can also vary the intensity and color of lighting elements (both artificial lighting and simulated skyboxes) in each scene, to simulate variations in indoor lighting and the time of the day. Assets (such as furniture and fruit) and larger structures such as walls and doors can be assigned a variety of colors and textures, sampled from sets of plausible colors and materials for each asset category. Together, the diversity of layouts, assets, placements, and lighting leads to an arbitrarily large set of environments – allowing PROCTHOR to scale orders of magnitude beyond the number of scenes currently supported by present-day simulators. In addition, PROCTHOR supports dynamic



Figure 1.1: We propose PROCTHOR, a framework to procedurally generate a large variety of diverse, interactable, and customizable houses.

material randomizations, whereby colors and materials of individual assets can be randomized each time an environment is loaded into memory for training. Importantly, in contrast to environments produced using 3D scans, scenes produced by PROCTHOR contain objects that both support a variety of different object states (*e.g.* open, closed, broken, *etc.*) and are fully interactive so that they can be physically manipulated by agents with robotic arms. We also present ARCHITECTHOR, a 3D artist-designed set of 10 high quality fully interactable houses, meant to be used as a test-only environment for research within household environments. In contrast to AI2-iTHOR (single rooms) and RoboTHOR (lesser visual diversity) environments, ARCHITECTHOR contains larger, diverse, and realistic houses.

We demonstrate the ease and effectiveness of PROCTHOR by sampling an environment of 10,000 houses (named PROCTHOR-10K), composed of diverse layouts ranging from small 1-room houses to larger 10-room houses. We train agents with very simple neural architectures (CNN+RNN) – *without* a depth sensor, and instead only employing RGB channels, with no explicit mapping and no human task supervision – on PROCTHOR-10K and produce state-of-the-art (SoTA) models on several navigation and interaction benchmarks. As of 10am PT on June 14th, 2022 we obtain (1) **RoboTHOR ObjectNav Challenge** [7] – 0-shot performance superior to the previous SoTA which uses RoboTHOR training scenes – with fine-tuning we obtain an 8.8 point improvement in SPL over the previous SoTA; (2) **Habitat ObjectNav Challenge 2022** [159] – top of the leaderboard results with a >3 point gain in SPL over the next best submission; (3) **1-phase Rearrangement Challenge 2022** [6] – top of the leaderboard results with Prop Fixed Strict improving from 0.19 to 0.245; (4) **AI2-iTHOR ObjectNav** – 0-shot numbers which already outperform a previous model that trains on AI2-iTHOR, with fine-tuning we achieve a success rate of 77.5%; (5) **ArmPointNav** [60] – 0-shot number that beats previous SoTA results when using RGB; and (6) **ArchitecTHOR ObjectNav** – a large success rate improvement from 18.5% to 31.4%. Finally, an ablation analysis clearly shows the advantages of scaling up from 10 to 100 to 1K and finally to 10K scenes and indicates that further improvements can be obtained by invoking PROCTHOR to produce even larger environments.

In summary, our contributions are (1) PROCTHOR, a framework that allows for the performant procedural generation of an unbounded number of diverse, fully-interactive, simulated environments, (2) ARCHITECTHOR, a new, 3D artist-designed set of houses for E-AI evaluation, and (3) SoTA results across six E-AI benchmarks covering manipulation and navigation tasks, including strong 0-shot results. PROCTHOR will be open-sourced and the code used in this work will be released.

1.2 Related Work

Embodied AI platforms. Various Embodied AI platforms have been developed over the past several years [127, 214, 223, 266, 77, 261]. These platforms target different design goals. AI2-THOR [127] and its variants (ManipulaTHOR [60] and RoboTHOR [50]) are built in the Unity game engine and focus on agent-object interactions, object state changes, and accurate physics simulation. Unlike AI2-THOR, Habitat [214] provides scenes constructed from 3D scans of houses, however, objects and scenes are not interactable. A more recent version, Habitat 2.0 [233], introduces object interactions at the

expense of being limited to one floorplan and synthetic scenes. iGibson [223] includes photo-realistic scenes, but with limited interactions such as pushing. iGibson 2.0 [137] extends iGibson by focusing on household tasks and object state changes in synthetic scenes and includes a virtual reality interface. ThreeDWorld [77] targets high-fidelity physics simulation such as liquid and deformable object simulation. VirtualHome [192] is designed for simulating human activities via programs. RLBench [105], RoboSuite [284] and Sapien [266] target fine-grained manipulation. The main advantage of PROCTHOR is that we can generate a diverse set of *interactive* scenes procedurally, enabling studies of data augmentation and large-scale training in the context of Embodied AI.

Large-scale datasets. Large-scale datasets have resulted in major breakthroughs in different domains such as image classification [53, 134], vision and language [25, 239], 3D understanding [21, 267], autonomous driving [17, 229], and robotic object manipulation [188, 167]. However, there are not many interactive large-scale datasets for Embodied AI research. PROCTHOR includes interactive houses generated procedurally. Hence, there are an arbitrarily large number of scenes in the framework. The closest works to ours are [198, 186, 141]. HM3D [198] is a recent framework that includes 1,000 scenes generated using 3D scans of real environments. PROCTHOR has a number of key distinctions: (1) unlike HM3D which includes static scenes, the scenes in PROCTHOR are interactive i.e., objects can move and change state, the lighting and texture of objects can change, and a physics engine determines the future states of the scenes; (2) it is challenging to scale up HM3D as it requires scanning a house and cleaning up the data, while we can procedurally generate more houses; (3) HM3D can be used only for navigation tasks (as there is no physics simulation and object interaction), while PROCTHOR can be used for tasks other than navigation. OpenRooms [141] is similar to HM3D in terms of the source of the data (3D scans) and dataset size. However, OpenRooms is interactive. OpenRooms is also confined to the set of scanned houses, and it takes a significant amount of time to annotate a new scene (e.g., labeling materials for one object takes 1 minute), while PROCTHOR does not suffer from these issues. Megaverse [186] is another large-scale Embodied AI platform that includes procedurally generated environments. Although it is impressive in terms of simulation speed, it includes only game-like environments with a simplified appearance. In contrast, PROCTHOR mimics real-world houses in terms of the complexity of appearance, physics, and object interactions.

Scene synthesis. Work on scene synthesis is typically broken down into generating floorplans [146, 153, 96, 260] and sampling object placement in rooms [68, 83, 276, 279, 30]. Our work aimed to generate diverse and semantically plausible houses using the best existing approaches or building on existing works in areas that were insufficient for our use case. Our floorplan generation process is adapted from [146, 153], which takes in a high-level specification of the rooms in a house and their connectivity constraints, and randomly generates floorplans satisfying these constraints. Our object placement is most similar to [279, 83, 276, 270, 24], where we iteratively place objects on floors, walls, and surfaces and use semantic asset groups to sample objects that co-occur (e.g. chairs next to tables). The modular generation process used in this work makes it easy to swap in and update any stage of our house generation pipeline with a better algorithm. In this work, we found the procedural generation approaches to be more reliable and flexible than the ones based on deep learning when adapting it to our custom object database and when generating more complex houses that were out of the distribution of static house datasets [71, 260, 142]. For a more detailed comparison, including a discussion of some of the limitations of deep learning approaches, please refer to the Appendix.

1.3 PROCTHOR

PROCTHOR is a framework to procedurally generate E-AI environments. It extends AI2-THOR and, thereby, inherits AI2-THOR’s large asset library, robotic agents, and accurate physics simulation. Just as in scenes painstakingly created by designers in AI2-THOR, environments in PROCTHOR are fully interactive and support navigation, object manipulation, and multi-agent interaction.

Fig. 1.2 shows a high-level schematic of the procedure used by PROCTHOR to generate a scene. Given a room specification (e.g. house with 1 bedroom + 1 bathroom), we use multi-stage conditional sampling to, iteratively, generate a floor plan, create an external wall structure, sample lighting, and doors, then sample assets including large, small and wall objects, pick colors and textures, and determine appropriate placements for assets within the scene. We refer the reader to the appendix for details regarding our procedural generation and sampling mechanism, but highlight five key characteristics of PROCTHOR: **Diversity**, **Interactivity**, **Customizability**, **Scale**, and **Efficiency**.

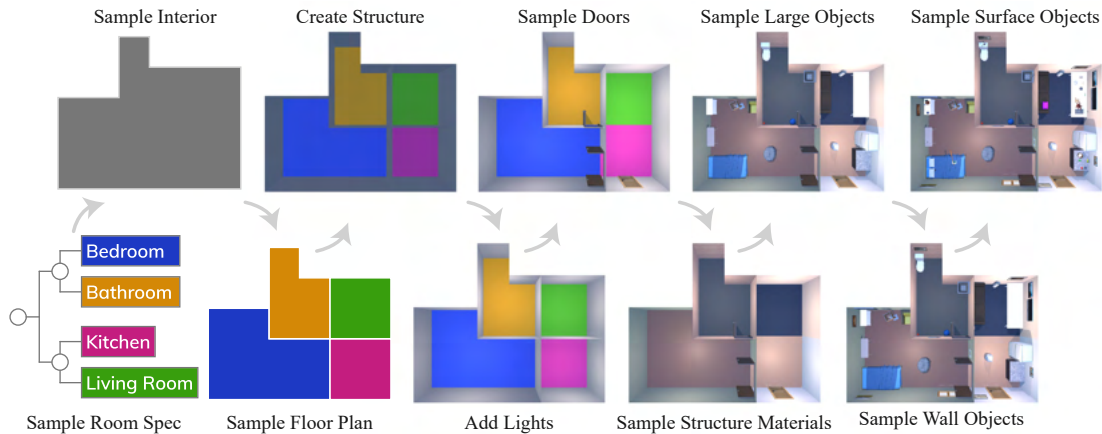


Figure 1.2: Procedurally generating a house using PROCTHOR.

Diversity. PROCTHOR enables the creation of rich and diverse environments. Mirroring the success of pre-training models with diverse data in the vision and NLP domains, we demonstrate the utility of this diversity on several E-AI tasks. Scenes in PROCTHOR exhibit diversity across several facets:

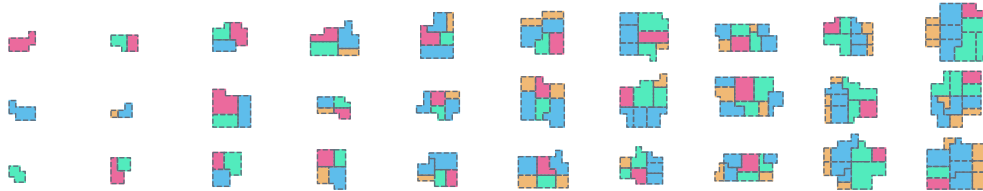


Figure 1.3: **Floorplan diversity.** Examples showing the diversity of the generated floorplans. Rooms in the house are colored by ■ Bedroom, ■ Bathroom, ■ Kitchen, and ■ Living Room.

Diversity of floor plans. Given a room specification, we first employ iterative boundary cutting to obtain an external scene layout (that can range from a simple rectangle to a complex polygon). The recursive layout generation algorithm by Lopes *et al.* [146] is then used to divide the scene into the desired rooms. Finally, we determine connectivity between rooms using a set of user-defined constraints. These procedures result in natural room layouts (e.g., bedrooms are often connected to adjoining bathrooms via a door, bathrooms more often have a single entrance, etc). As exemplified in Fig. 1.3, PROCTHOR generates hugely diverse floor plans using this procedure.



Figure 1.4: **Object diversity.** A subset of instances for four object categories.

Diversity of assets. PROCTOR populates scenes with small and large assets from its database of 1633 household assets across 108 categories (examples in Fig. 1.4). While many assets are inherited from AI2-THOR, we also introduce new assets such as windows, doors, and countertops, hand-designed by 3D graphic designers. Asset instances are split into train/val/test subsets and are interactable, i.e. objects can be picked and placed within the scenes, some objects have multiple states (e.g. a light can be on or off) and several objects consists of parts with rigid body motions (e.g. door on a microwave).



Figure 1.5: **Material augmentation.** Different materials for objects and structural elements.

Diversity of materials. Walls can have two kinds of materials – one of 40 solid (and popular) colors or one of 122 wall textures such as brick and tile. We also provide 55 floor materials. The ceiling material for the entire house is sampled from the set of wall materials. PROCTOR also provides the ability to randomize materials of objects. Materials are only randomized within categories, which ensures objects still look and behave like the class they represent.



Figure 1.6: **Object placement.** Four examples of object placement within the same room layout.

Diversity of object placements. Asset categories have several soft annotations that help place them realistically within a house. These include room assignments (e.g. couch in a living room but not a bathroom) and location assignments (e.g. fridge along a wall, TV not on the floor). We also develop the notion of a Semantic Asset Group (SAG) – groups of assets that typically co-occur (e.g. dining table with four chairs) and thus must be sampled and placed using dependent sampling. Given a layout, individual assets and SAGs that lie on the floor are sampled and placed iteratively, ensuring that rooms continue

to have adequate floor space for agents to navigate and manipulate objects. Then wall objects such as windows and paintings get placed, and finally, surface objects (ones found on top of other assets) are placed (*e.g.* cups on the kitchen counter). This sampling allows for a large and diverse set of object choices and placements within any layout. Fig. 1.6 shows such variations.



Figure 1.7: **Lighting variation.** Morning, dusk, and night lighting for an example scene.

Diversity of lighting. PROCTOR supports a single directional light (analogous to the sun) and several point lights (analogous to lightbulbs). Varying the color, intensity, and placement of these sources allows us to simulate different artificial lighting, typically observed in houses, and also at different times of the day. Lighting has a significant effect on the rendered images as seen in Fig. 1.7.



Figure 1.8: **Interactivity.** Object states can change (*e.g.*, the laptop or the lamp in the left panel), and the agents can interact with objects and other agents (middle and right panels).

Interactivity. A key property of PROCTOR is the ability to interact with objects to change their location or state (Fig. 1.8). This capability is fundamental to many Embodied AI tasks. Datasets like HM3D [198] that are created from static 3D scans do not possess this capability. PROCTOR supports agents with arms capable of manipulating objects and interacting with each other.



Figure 1.9: **Customizability.** PROCTOR can be used to construct custom scene types such as classrooms, libraries, and offices.

Customizability. PROCTOR supports many room, asset, material, and lighting specifications. With a few simple lines of specification, one can easily generate customized environments of interest. Fig. 1.9 shows examples of such varied scenes (classroom, library, and office).

Scale and Efficiency. PROCTOR currently uses 16 different scene specifications to seed the scene generation process. These can result in over 100 billion layouts. PROCTOR uses 18 different Semantic Asset groups and 1633 assets. These can result in roughly 20 million unique asset groups. Each of these assets can be placed in numerous locations. In addition, each house gets scaled and uses a variety of lighting. This diversity of layouts, assets, materials, placements, and lighting enables the generation of *arbitrarily large* sets of houses – either statically generated and stored as a dataset or dynamically generated at each iteration of training. Scenes are efficiently represented in a JSON specification and are loaded into AI2-THOR at runtime, making the memory overhead of storing houses incredibly efficient. Scene generation is fully automatic and fast and PROCTOR provides high framerates for training E-AI models (see Sec. 1.4 for details).

1.4 PROCTOR-10K

We demonstrate the power and potential of PROCTOR using a sampled set of 10,000 fully interactive houses obtained by the procedural generation process described in Section 1.3 – which we label PROCTOR-10K. An additional set of 1,000 validation and 1,000 testing houses are available for evaluation. Asset splits across train/val/test are detailed in the Appendix. All houses are fully navigable, allowing an agent to traverse through each room without any interaction. In terms of scale, PROCTOR-10K is one of the largest sets of interactive home environments for Embodied AI – as a comparison, AI2-iTHOR [127] includes 120 scenes, RoboTHOR [50] has 89 scenes, iGibson [223] has 15 scenes, Habitat Matterport 3D [198] has 1,000 static (non-interactive) scenes, and Habitat 2.0 [233] has 105 scene layouts. Scaling beyond 10K houses is straightforward and inexpensive. This set of 10K houses was generated in 1 hour on a local workstation with 4 NVIDIA RTX A5000 GPUs. Fig. 1.11 shows examples of ego-centric and top-down views of houses present in PROCTOR-10K.

Scene statistics. Houses in PROCTOR-10K are generated using 16 different room specifications. An example room spec is: *A house with 1 bedroom connected to 1 bathroom, 1 kitchen, and 1 living room*

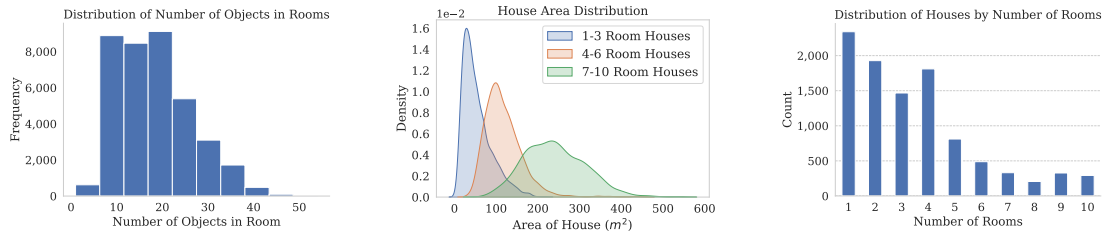


Figure 1.10: **PROCTOR-10K statistics.** *Left:* distribution of the number of objects in each room; *Middle:* distribution of the area of each house, bucketed into small, medium, and large houses; *Right:* bar plot showing the distribution over the number of rooms that make up each house.



Figure 1.11: **Example scenes** in PROCTOR-10K with top-down and an egocentric view.

Compute	Navigation FPS		Isolated Interaction FPS		Environment Query FPS	
	Small	Large	Small	Large	Small	Large
8 GPUs	8,599±359	3,208±127	6,488±250	2,861±107	480,205±19,684	433,587±18,729
1 GPU	1,427±74	6,280±40	1,265±71	597±37	160,622±2,846	157,567±2,689
1 Process	240±69	115±19	180±42	93±15	14,825±199	14,916±186

Table 1.1: **Rendering speed.** Benchmarking FPS for navigation (*e.g.* moving/rotating), interaction (*e.g.* pushing an object), and querying the environment for data (*e.g.* checking the dimensions of the agent). We report FPS for Small and Large houses. See Appendix for details.

and is visualized in Fig. 1.2. Houses in this dataset have as few as 1 room and as many as 10. Fig. 1.10 shows the distribution of areas (middle) and the number of rooms (right) of these generated houses. Our use of room specifications enables us to change the distribution of the size and complexity of houses fairly easily. PROCTOR-10K encompasses a wider spectrum of scenes than AI2-iTHOR [127] and ROBOTHOR [50] (biased towards room-sized scenes) and Gibson [264] and HM3D [198] (biased towards large houses).

Rooms in each of these houses contain objects from 95 different categories including common household objects such as fridges, countertops, beds, toilets, and house plants, and structure objects such as doorways and windows. Fig. 1.10 (left) shows the distribution of the number of objects per room per house, which shows that houses in PROCTOR-10K are well populated. They also contain objects sampled via 18 different Semantic Asset groups. Examples of Semantic asset groups (SAG) are a *Dining Table with 4 Chairs* or *Bed with 2 Pillows*. Given our large asset library and SAGs, we can create 19.3 million combinations of group instantiations.

Rendering speed. A crucial requirement for large-scale training is high rendering speed since the training algorithms require millions of iterations to converge. Table 1.1 shows these statistics. Experiments were run on a server with 8 NVIDIA Quadro RTX 8000 GPUs. For the 1 GPU experiments, we use 15 processes and for the 8 GPU experiments, we use 120 processes, evenly distributed across the GPUs. PROCTOR provides framerates comparable to iTHOR and RoboTHOR environments in spite of having larger houses (See Appendix for details), rendering it fast enough for training large models for hundreds of millions of steps in a reasonable amount of time.

1.5 ArchitecTHOR



Figure 1.12: Top-down images of ARCHITECTOR validation houses.

In order to test if models trained on ProcTHOR can generalize to real-world floorplans and object placements, a test set of houses was needed. Neither iTHOR (single room scenes) nor RoboTHOR (dorm-sized maze-styled scenes) contain scenes that are representative of real-world homes. Therefore, we worked with professional 3D artists to create ArchitecTHOR, which contains 10 evaluation houses (5 val, 5 test) that mimic the style of real-world homes. ArchitecTHOR val houses contain between 4-8 rooms, 121 ± 26 objects per house, and a typical floor size of $111 \pm 26 m^2$. By comparison, PROCTOR-10K

houses have a much higher variance, with between 1-10 rooms, 76 ± 48 objects per house, and a typical floor size of $96 \pm 74 \text{ m}^2$.

1.6 Experiments

Tasks. We now present results for models pre-trained on PROCTHOR-10K on several navigation and manipulation benchmarks to demonstrate the benefits of large-scale training. We consider ObjectNav (navigation towards a specific object category) in PROCTHOR, ARCHITECTHOR, RoboTHOR [50], HM3D [198], and AI2-iTHOR [127]. We also consider two manipulation-based tasks: ArmPointNav [60] and 1-phase Room Rearrangement [249]. In ArmPointNav, the agent moves an object using a robotic arm from a source location to a destination location specified in the 3D coordinate frame. In Room Rearrangement, the goal is to move objects or change their state to reach a target scene state.

Models. Our models for all tasks consist of a CNN to encode visual information and a GRU to capture temporal information. We deliberately use a simple architecture across all tasks to show the benefits of large-scale training. Our ObjectNav and Rearrangement models use the CLIP-based architectures of [117]. Our ArmPointNav model uses a simpler visual encoder with 3 convolutional layers; we found this more effective than the CLIP encoder. All models are trained with the AllenAct [251] framework, see the Appendix for training details.

Results. We present results in two settings: zero-shot and after fine-tuning on the training scenes provided by the downstream benchmark. Zero-shot experiments show us how well models trained on PROCTHOR generalize to new environments, whereas fine-tuning experiments tell us if representations learned from PROCTHOR can serve as a good initialization for quick tuning. For all experiments, we use only RGB images (no depth and other modalities is used).

Zero-shot is particularly challenging since other environments have different appearance statistics, layouts, and object distributions compared to PROCTHOR. ARCHITECTHOR and AI2-iTHOR [127] are high-fidelity artist-designed scenes with high-quality shadows and lighting. HM3D is constructed from 3D scans of houses which can differ quite a bit from synthetic environments. RoboTHOR [50] houses use wall panels and floors with very specific textures.

Zero-shot transfer results. Models trained only on PROCTHOR and evaluated 0-shot outperform previous SoTA models on 3 benchmarks (see Table 1.2). These strong results suggest that models generalize to not only unseen objects and scenes, but also new appearance and layout statistics.

Fine-tuning results. Further fine-tuning of the model using each benchmark’s training data, achieves state-of-the-art results on all benchmarks (refer to *fine-tune* rows of Table 1.2). Notably, our model is ranked first on three public leaderboards as of 10am PT, June 14th 2022: Habitat 2022 ObjectNav challenge, AI2-THOR Rearrangement 2022 challenge, and RoboTHOR ObjectNav challenge. It should be noted that our model achieves these results using a very simple architecture and only RGB images. Other techniques typically use more complex architectures that include mapping or visual odometry modules and use additional perception sensors such as depth images.

Scale ablation. To evaluate the effect of scale we train the models on 10, 100, 1k, and 10k houses. Here, we do not use any material augmentations. As shown in Table 1.3, the performance improves as we use more houses for training, demonstrating the benefits of large-scale data for E-AI tasks.

1.7 Conclusion

We propose PROCTHOR to procedurally generate *arbitrarily large* sets of interactive, physics-enabled houses for Embodied AI research. We pre-train simple models on 10k generated houses and show SOTA results across 6 embodied tasks with strong 0-shot results.

Task	Benchmark	Method	Metrics	
			Success	SPL
ObjectNav	RoboTHOR Challenge	EmbCLIP [117] ^a	47.0%	0.200
		ProcTHOR 0-shot	55.0%	0.237
		ProcTHOR + fine-tune	65.2%	0.288
ObjectNav	Habitat Challenge (2022) <i>HM3D-Semantics</i>	MLNLC ^c	52.0%	0.280
		FusionNav (AIRI) ^c	54.0%	0.270
		ProcTHOR 0-shot	9.00%	0.055
		ProcTHOR + fine-tune	53.0%	0.270
		ProcTHOR + Large ^d + 0-shot	13.2%	0.077
		ProcTHOR + Large ^d + fine-tune	54.4%	0.318
ObjectNav	AI2-iTHOR	EmbCLIP [117] ^b	68.4%	0.516
		ProcTHOR 0-shot	75.7%	0.644
		ProcTHOR + fine-tune	77.5%	0.621
ObjectNav	ARCHITECTHOR	EmbCLIP [117] ^b	18.5%	0.118
		ProcTHOR	31.4%	0.195
Rearrangement	AI2-THOR Challenge <i>1-phase (2022)</i>	EmbCLIP [117]	7.10%	0.190
		ProcTHOR 0-shot	3.80%	0.156
		ProcTHOR + fine-tune	7.40%	0.245
ArmPointNav	ManipulaTHOR	iTHOR-SimpleConv [60] ^e	29.2%	73.4
		ProcTHOR 0-shot	37.9%	74.8

Table 1.2: Results for models trained on ProcTHOR and evaluated 0-shot and with fine-tuning on several E-AI benchmarks. For each benchmark we also compare to the relevant baselines (previous SoTA or leaderboard submissions where applicable). ^aEmbCLIP [117] trained on ROBOTHOR, ^bEmbCLIP [117] trained on AI2-iTHOR, ^csubmission on the Habitat 2022 ObjectNav leaderboard [159]. ^dFor HM3D we present results when pretraining using the EmbCLIP architecture (which uses CLIP-pretrained ResNet50) as well as with a “Large” model which uses a larger CLIP backbone CNN as well as a wider RNN, see supplement for details. ^euses the model from [60] but retrains on the complete iTHOR data with RGB inputs. 0-shot results, whereby models are pre-trained on PROCTOR-10K and do not use any training data from the benchmark that they are evaluated on.

# HOUSES	ARCHITECTHOR Test		ROBOTHOR Test (0-Shot)		HM3D Valid (0-Shot)		AI2-iTHOR Test (0-Shot)	
	SPL	SR	SPL	SR	SPL	SR	SPL	SR
10 Houses	0.077	11.3%	0.040	8.53%	0.007	1.60%	0.249	28.7%
100 Houses	0.102	18.6%	0.076	20.9%	0.050	10.4%	0.352	42.0%
1,000 Houses	0.122	17.2%	0.157	33.1%	0.027	4.65%	0.456	53.0%
10,000 Houses	0.185	27.0%	0.210	44.5%	0.060	9.70%	0.554	64.9%

Table 1.3: Ablation study to evaluate the effect of the number of training houses. Each model is trained to 80% success during training. Test performance increases with the number of training houses.

Chapter 2

Objaverse

2.1 Abstract

Massive data corpora like WebText, Wikipedia, Conceptual Captions, WebImageText, and LAION have propelled recent dramatic progress in AI. Large neural models trained on such datasets produce impressive results and top many of today’s benchmarks. A notable omission within this family of large-scale datasets is 3D data. Despite considerable interest and potential applications in 3D vision, datasets of high-fidelity 3D models continue to be mid-sized with limited diversity of object categories. Addressing this gap, we present Objaverse 1.0, a large dataset of objects with 800K+ (and growing) 3D models with descriptive captions, tags, and animations. Objaverse improves upon present day 3D repositories in terms of scale, number of categories, and in the visual diversity of instances within a category. We demonstrate the large potential of Objaverse via four diverse applications: training generative 3D models, improving tail category segmentation on the LVIS benchmark, training open-vocabulary object-navigation models for Embodied AI, and creating a new benchmark for robustness analysis of vision models. Objaverse can open new directions for research and enable new applications across the field of AI.

2.2 Introduction

Massive datasets have enabled and driven rapid progress in AI. Language corpora on the web led to large language models like GPT-3 [15]; paired image and text datasets like Conceptual Captions [222] led to vision-and-language pretrained models like ViLBERT [147]; YouTube video datasets led to video capable models like Merlot-Reserve [277]; and massive multimodal datasets like WebImageText [227] and LAION [217, 216] led to models like CLIP [195] and StableDiffusion [207]. These leaps in dataset scale and diversity were triggered by moving from manually curated datasets to harnessing the power of the web and its creative content.

In contrast to the datasets described above, the size of the datasets we are feeding to our data-hungry deep learning models in many other areas of research is simply not comparable. For instance, the number of 3D assets used in training generative 3D models is, maximally, on the order of thousands [80] and the simulators used to train embodied AI models typically have only between a few dozen to a thousand unique scenes [125, 233, 202, 136]. The startling advances brought about by developing large-scale datasets for images, videos, and natural language, demand that an equivalent dataset be built for 3D assets.

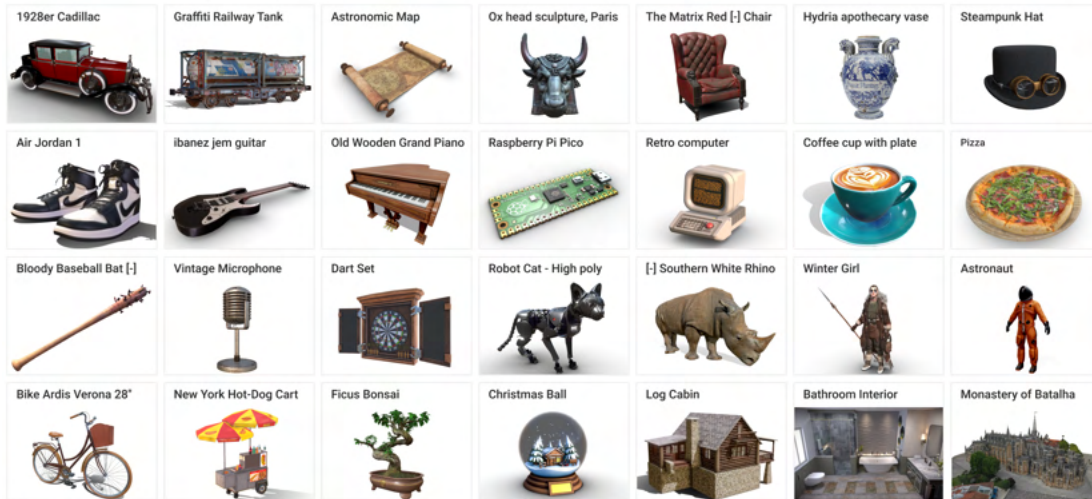


Figure 2.1: Example instances from our large-scale 3D asset dataset OBJAVERSE. OBJAVERSE 3D assets are semantically diverse, high-quality, and paired with natural-language descriptions.

We present OBJAVERSE 1.0, a large scale corpus of high-quality, richly annotated, 3D objects; see Fig. 2.1. Objects in our dataset are free to use¹ and sourced from Sketchfab, a leading online platform for managing, viewing, and distributing 3D models. In total, OBJAVERSE contains over **800K** 3D assets designed by over **150K** artists which makes this data large and diversely sourced. Assets not only belong to varied categories like animals, humans, and vehicles, but also include interiors and exteriors of large spaces that can be used, *e.g.*, to train embodied agents. OBJAVERSE is a universe of rich 3D data with detailed metadata that can support many different annotations to enable new applications. With this remarkable increase in scale, we see an incredible opportunity for OBJAVERSE to impact research progress across domains. In this work, we provide promising results to answer three questions.

Can 3D vision benefit from a large-scale dataset? First, as a 3D asset resource, OBJAVERSE can support the exciting field of 3D generative modeling. We use data extracted from OBJAVERSE to train generative models for single and multiple categories using GET3D [80] and find that we are able to generate high-quality objects. Moreover, we find that our generated objects are found by human annotators to be more diverse than those generated by a model trained on ShapeNet objects in 91% of cases.

Can the diversity of 3D models help improve classical 2D vision task performance? To answer this question, we use the diversity of OBJAVERSE to improve the performance of long tail instance segmentation models. Instance segmentation data can be expensive to obtain owing to the cost of annotating contours around objects. The recent LVIS dataset contains segmentation annotations for 1,230 categories but the task remains very challenging for present day models, particularly on tail categories that have few examples. We show that increasing the volume of data by leveraging a simple Copy+Paste augmentation method with OBJAVERSE assets can improve the performance of state-of-the-art segmentation methods.

¹Creative Commons license

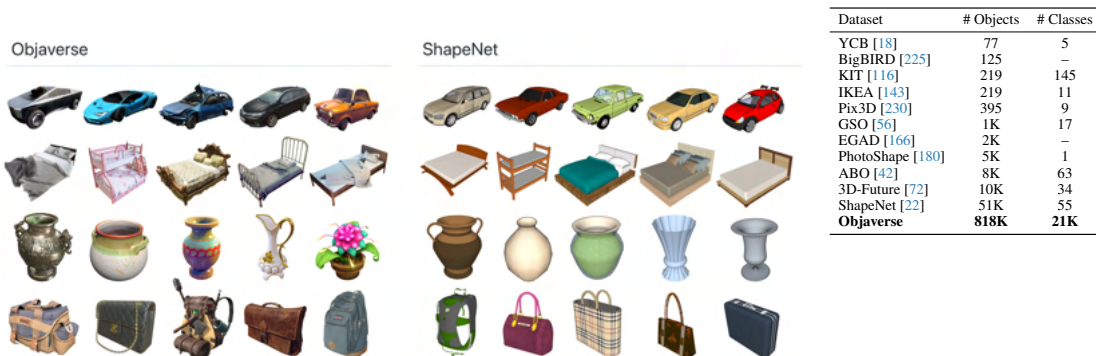


Figure 2.2: Comparison between OBJAVERSE and existing 3D object datasets. (Left:) Visual comparison of instances from OBJAVERSE and ShapeNet for the categories of CAR, BED, VASE, and BAG. OBJAVERSE instances are substantially more diverse since objects can come from many 3D content creation platforms, whereas ShapeNet models look more similar and all come from SketchUp, a 3D modeling platform built for simple architectural modeling. (Right:) Scale comparison table between existing 3D object datasets.

We also use OBJAVERSE to build a benchmark for evaluating the robustness of state-of-the-art visual classification models to perspective shifts. We render objects in OBJAVERSE from random orientations, which is how one might expect to see them in the real world, and test the ability of CLIP-style visual backbones to correctly classify these images. Our experiments show that current state-of-the-art models’ performance degrades dramatically in this setting when viewing objects from arbitrary views. OBJAVERSE allows us to build benchmarks to test (and potentially train) for orientation robustness for a long tail distribution of asset categories. Building such benchmarks is made uniquely possible by the scale and diversity of 3D assets in OBJAVERSE. This would simply not be feasible to create in the real world nor can they be generated from existing 2D images.

Can a large-scale 3D dataset help us train performant embodied agents? We use assets in OBJAVERSE to populate procedurally generated simulated environments in ProcTHOR [52] that are used to train Embodied AI agents. This results in an orders of magnitude increase in the number of unique assets available for use in ProcTHOR scenes (previously limited to AI2-THOR’s [125] asset library of a few thousand unique instances each assigned to one of 108 object categories). Using OBJAVERSE populated scenes enables open vocabulary object navigation from any text description. In this paper, we provide quantitative results for navigating to 1.1K semantic object categories, roughly a 50x increase.

These findings represent just a small fraction of what can be accomplished using OBJAVERSE. We are excited to see how the research community will leverage OBJAVERSE to enable fast and exciting progress in 2D and 3D computer vision applications and beyond.

2.3 Related Work

Large scale datasets. Scaling the size and scope of training datasets has widely been demonstrated to be an effective avenue of improvement for model performance. In computer vision, the adoption of early large scale datasets such as Imagenet[210, 54] and MS-COCO[145] has dramatically accelerated progress on a variety of tasks including classification, object detection, captioning, and more. Ever since, the diversity and scale of datasets have continued to grow. YFCC100M is a dataset of 99.2M images and 800K videos[238]. OpenImages[133] is a large scale dataset of 9M images that contains labeled subsets bounding boxes, visual relationships, segmentation masks, localized narratives, and categorical annotations. Massive web-scraped datasets containing image-text pairs such as Conceptual

Captions[222], WIT[227], and LAION[217, 216] have seen increased popularity recently as they have been used to train impressive models for vision-language representation learning[195, 98, 106], text-to-image generation[201, 199, 207, 106], and vision-language multitasking[38, 234, 247, 36].

3D datasets. Current large-scale 2D image datasets offer three crucial components that benefit learning: scale, diversity, and realism. Ideally, models that reason about 3D objects should have access to datasets that meet these same criteria. However, of the numerous 3D object datasets that currently exist, none are able to excel in all three categories to the same degree as their 2D counterparts. Datasets such as KIT[116], YCB[18], BigBIRD[225], IKEA[143], and Pix3D[230] provide image-calibrated models over a diverse set of household objects, but severely lack in scale with only a few hundred objects at most. EGAD[166] procedurally generates 2K objects for grasping, but produces objects that are not that realistic or diverse. Slightly larger datasets of photo-realistic objects include GSO[56], PhotoShape[180], ABO[42] and 3D-Future[72], and ShapeNet[22] with object counts in the tens of thousands, see Fig. 2.2 for comparisons between OBJAVERSE and these datasets. Datasets for CAD models, such as ModelNet[262] and DeepCAD[259], and ABC[122] do not include textures or materials, which limits their ability to represent objects that could plausibly be found in the real world. Datasets of scanned 3D objects and environments are valuable for real-world understanding[45, 46, 40, 135], but are quite small and limited. In addition to containing numerous artist designed objects, OBJAVERSE contains many scanned assets, making it a useful source of data for learning from real-world distributions.

While rapid progress has been made in developing datasets that combine image and text, in contrast, only a few datasets that pair language and 3D data exist. Text2Shape[35] released a dataset of 15,038 chairs and tables from ShapeNet each with around 5 text captions, giving 75,344 total text-shape pairs. ShapeGlot[1] released the CiC (Chairs in Context) dataset which contains 4,511 chairs from ShapeNet along with 78,789 descriptive utterances generated from a referential game. Due to the small scale and limited diversity of these datasets, current SoTA text-to-3D models[160, 189, 100] forgo the use of 3D datasets entirely and instead rely on 2D image-text supervision.

2.4 Objaverse

OBJAVERSE is a massive annotated 3D dataset that can be used to enable research in a wide range of areas across computer vision. The objects are sourced from Sketchfab, an online 3D marketplace where users can upload and share models for both free and commercial use. Objects selected for OBJAVERSE have a distributable Creative Commons license and were obtained using Sketchfab’s public API. Aside from licensing consideration, models marked as restricted due to objectionable or adult thematic content were excluded from the dataset.

Model metadata. OBJAVERSE objects inherit a set of foundational annotations supplied by their creator when uploaded to Sketchfab. Figure 2.4 shows an example of the metadata available for each model. The metadata includes a name, assignments to a set of fixed categories, a set of unrestricted tags, and a natural language description.

OBJAVERSE-LVIS. While OBJAVERSE metadata contains a great deal of information about objects, Sketchfab’s existing categorization scheme covers only 18 categories, too coarse for most applications. Object names, categories, and tags provide multiple potential categorizations at varying levels of specificity and with some inherent noise. However, for many existing computer vision tasks, it is useful to assign objects to a single category drawn from a predetermined set of the right size and level of semantic granularity.

We choose the categories from the LVIS dataset [87] for categorizing a long-tail subset of objects in OBJAVERSE. We construct a 47K LVIS categorized object subset, called OBJAVERSE-LVIS, comprised of objects uniquely assigned to one of 1156 LVIS categories. We perform these assignments by first selecting 500 candidate objects per category using a combination of predictions from a CLIP classification

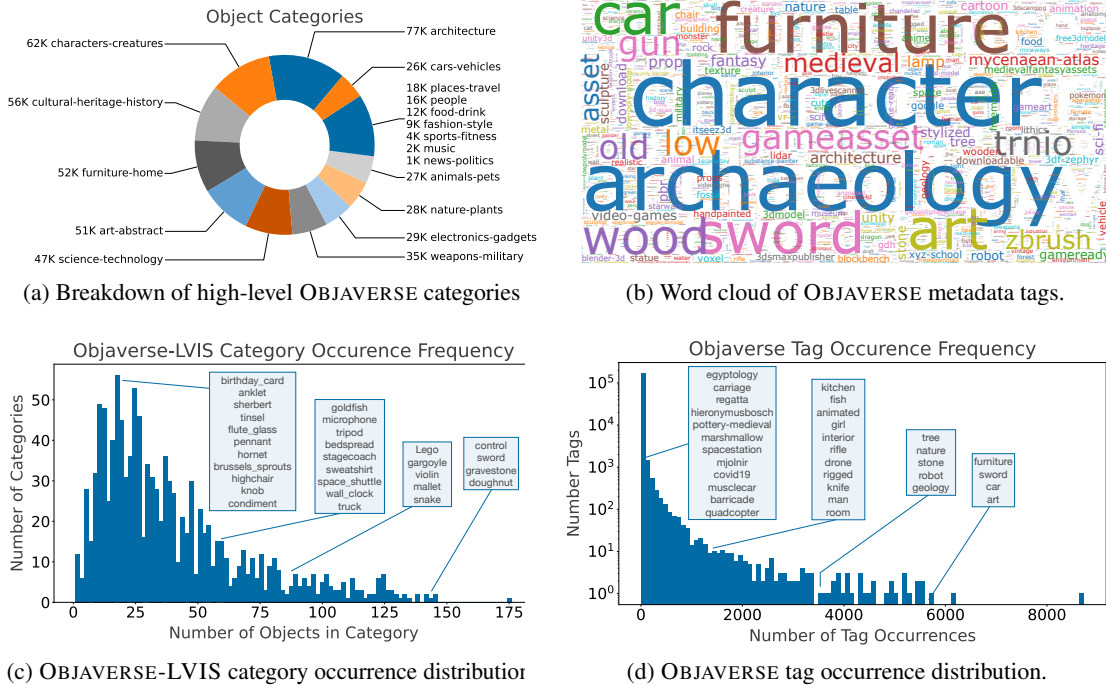


Figure 2.3: **OBJAVERSE statistics.** (a) All 18 high-level categories present in OBJAVERSE’s metadata with their corresponding number of occurrences. The relative share of most popular categories are evenly split, with a small number of less frequently categories. (b) A sample of several thousand popular object tags found in OBJAVERSE log-scaled by their frequency. (c) A histogram of fine-grained OBJAVERSE-LVIS categories with representative members from several bins highlighted. (d) A histogram of OBJAVERSE tags with representative members from several bins highlighted (note y-axis log scale). Tags from the low-occurrence side of the distribution correspond to unique objects that, taken individually, are rarely seen in the world. Frequently used tags like "furniture" and "car" reflect their real-world normalcy, but the high frequency of assets like "sword" diverge from their real-world counterparts.

model and candidates suggested by terms in their metadata. This combined pool contains objects visually resembling the target category (from the CLIP features of their thumbnail images) that might have missing metadata, as well as visually unusual instances of a category that are accurately named or tagged. These 250k candidate objects were then manually filtered and their assigned categories verified by crowdworkers. Since we only presented 500 object candidates per class, many popular categories, such as chair or car, have substantially more objects that could be included in OBJAVERSE-LVIS with future annotations.

Animated objects and rigged characters. OBJAVERSE includes 44K animated objects and over 63K objects self-categorized as characters. Examples of animations include fridge doors opening, animals running, and the hands on a clock moving. Rigged characters can be set up for animation and rendering, and may often come annotated with bone mappings. The vast scale of animations available in OBJAVERSE can support a wide range of research in temporal 3D learning, such as building text-based animation generative models [237], representing object changes over time with NERFs [193, 181], and temporal self-supervised learning via. future frame prediction [277, 99].

Articulated objects. Decomposing 3D objects into parts has led to a flurry of research in the past few years, including work in learning robotic grasping policies [271, 266], 3D semantic segmentation [164],

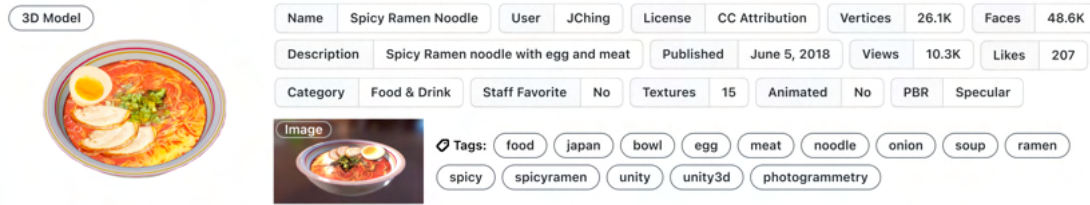


Figure 2.4: An example of metadata available for each object in OBJAVERSE. Each uploaded object has a 3D model, user-selected rendered thumbnail image, name, description, tags, category, and stats, among additional metadata.

and shape generation [163]. Since many objects in OBJAVERSE were uploaded by artists, the objects often come separated into parts. Figure 2.5 shows an example, where a chair is separated by its backrest, wheels, and legs, among many smaller parts.

Exteriors. Photogrammetry and NERF advances have enabled the commercialization of capturing high-quality 3D objects of large exteriors by taking pictures [236, 268]. In OBJAVERSE, there are a large number of scanned buildings, cities, and stadiums. Figure 2.5 shows an example of a 3D object of NYC’s skyline captured through a scan.

OBJAVERSE-Interiors. There are 16K+ interior scenes in OBJAVERSE, including houses, classrooms, and offices. The scenes often have multiple floors, many types of rooms, and are densely populated with objects from human input. Objects in the scenes are separable into parts, which allows them to be usable for interactive robotics, embodied AI, and scene synthesis. To put the scale of OBJAVERSE-Interiors in perspective, the number of scenes in OBJAVERSE-Interiors is significantly larger than the 400 or so existing hand-built interactive embodied AI scenes [125, 76, 136, 233].

Visual styles. Objects in the world can be constructed in many styles and often differ in style based on the time-period, geographic location, and artist’s style. OBJAVERSE objects cover a vast set of visual styles, including 3D scans, 3D modeled objects from virtually any platform, point clouds, and photo-realism via physically based rendering (PBR) [187]. Moreover, instances of objects often appear with many styles, which is critical for training and evaluating robust computer vision models [195]. Figure 2.5 shows examples of chairs in OBJAVERSE in many different styles, including Gothic, modern, Victorian, cartoon, and abstract.

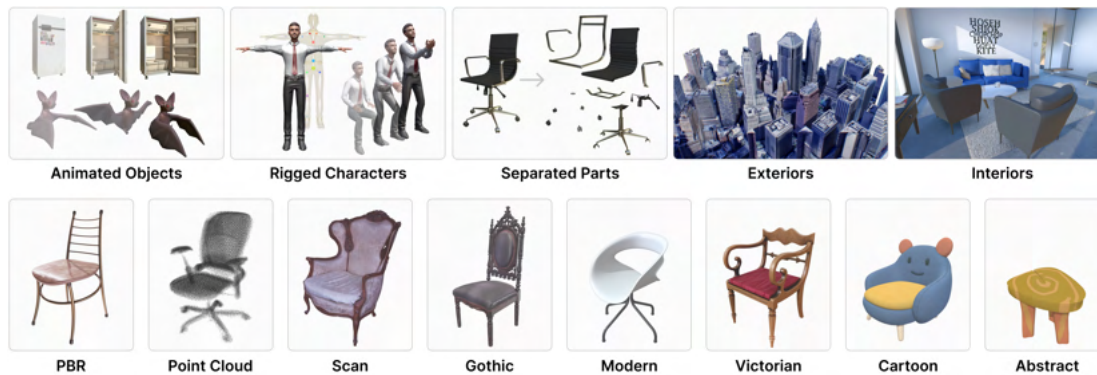
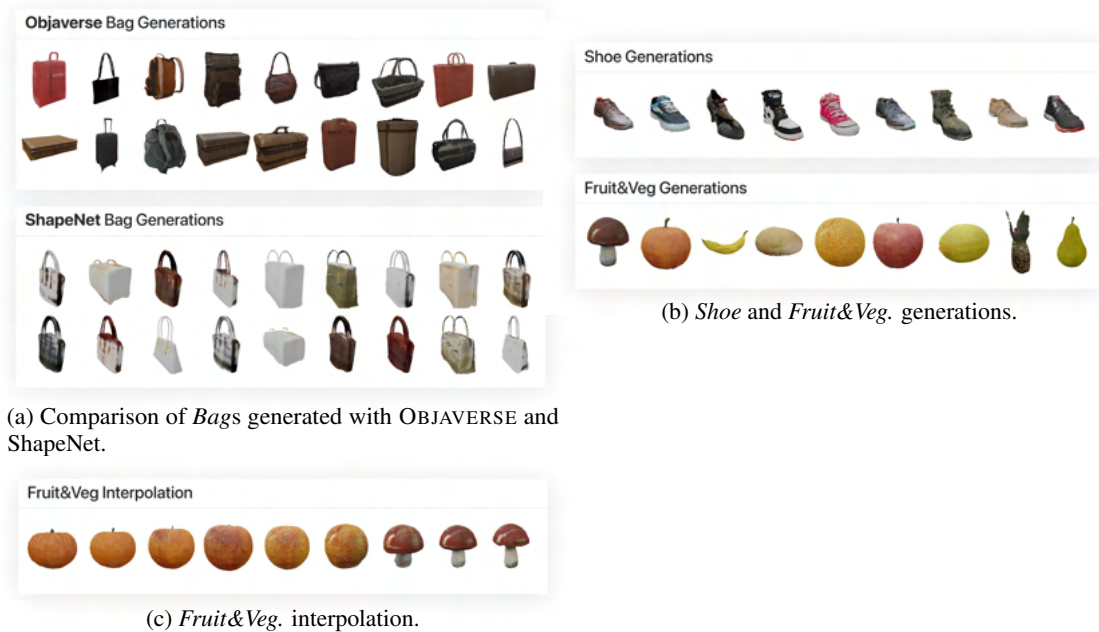


Figure 2.5: Highlights of the visual diversity of objects that appear in OBJAVERSE, including animated objects, rigged (body-part annotated) characters, models separatable into parts, exterior environments, interior environments, and a wide range visual styles.



(a) Comparison of *Bags* generated with OBJAVERSE and ShapeNet.

(b) *Shoe* and *Fruit&Veg.* generations.

(c) *Fruit&Veg.* interpolation.

Figure 2.6: (a) Example GET3D *Bag* object generations using OBJAVERSE and ShapeNet models for training. (b) Additional *Shoe* and *Fruit&Veg* generations from OBJAVERSE models. (c) models generated when interpolating between two, randomly sampled, latent encodings with our trained Fruit&Veg. model; what appears to be a pumpkin smoothly transforms into a mushroom.

Statistics. OBJAVERSE 1.0 includes 818K 3D objects, designed by 160K artists. There are $>2.35\text{M}$ tags on the objects, with $>170\text{K}$ of them being unique. We estimate that the objects have coverage for nearly 21K WordNet entities [162] (see appendix for details). Objects were uploaded between 2012 and 2022, with over 200K objects uploaded just in 2021. Figure 2.3 visualizes several statistics of the dataset, including the breakdown of objects into their self-assigned Sketchfab categories, a word cloud over the tags, a frequency plot of the tags, and the number of objects in OBJAVERSE-LVIS categories.

2.5 Applications

In this section, we present 4 initial distinct applications of OBJAVERSE, including 3D generative modeling, instance segmentation with CP3D, open-vocabulary ObjectNav, and analyze robustness in computer vision models.

2.5.1 3D Generative Modeling

3D generative modeling has shown much improvement recently with models such as GET3D [80] delivering impressive high quality results with rich geometric details. GET3D is trained to generate 3D textured meshes for a category and produces impressive 3D objects for categories like *Car*, *Chair*, and *Motorcycle* using data from ShapeNet [22]. OBJAVERSE contains 3D models for many diverse categories including tail categories which are not represented in other datasets. It also contains diverse and realistic object instances per category. This scale and diversity can be used to train large vocabulary and high quality 3D generative models. In this work, we showcase the potential of this data as follows. We choose three categories of objects, *Shoe*, *Bag*, and *Fruit&Veg*, and subsample objects from OBJAVERSE to create

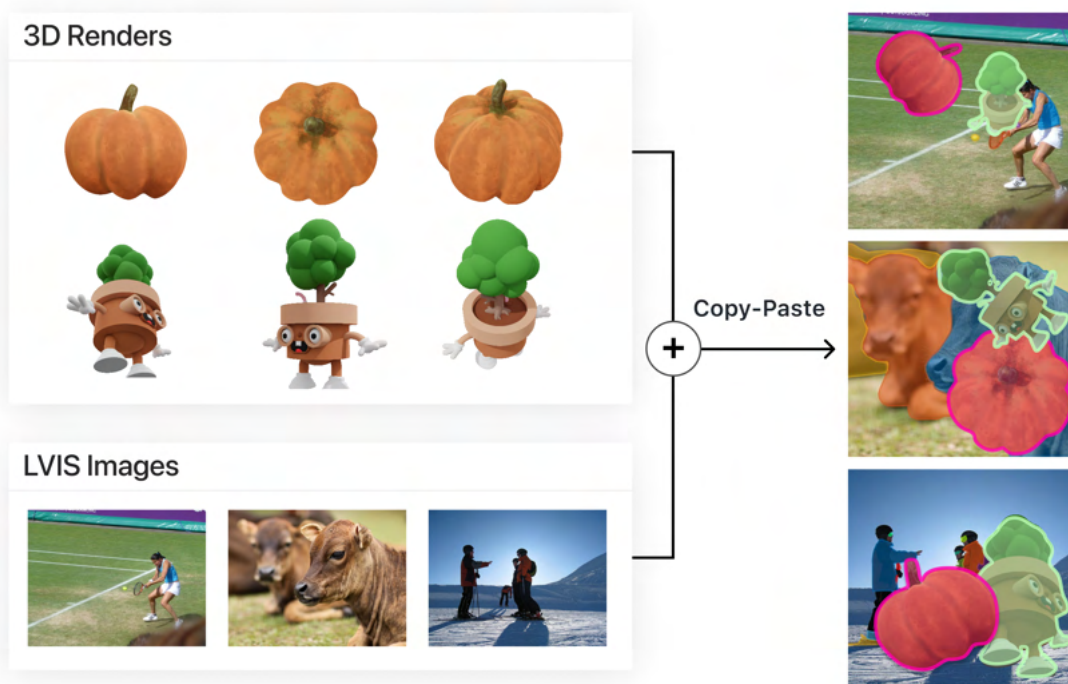


Figure 2.7: An illustration of CP3D (copy-paste 3D) for segmentation augmentation. We render 3D objects from multiple views and paste them over LVIS training images.

three separate datasets containing, respectively, 143 shoes, 816 bags, and 571 fruits & vegetables (116 apples, 112 gourds, 92 mushrooms, 68 bananas, 52 oranges, 52 pears, 31 potatoes 24 lemons, and 24 pineapples). For comparison, we also train a GET3D model on the set of 83 bags from the ShapeNet dataset. Fig. 2.6 shows a collection of 3D objects generated by our trained GET3D models. Qualitatively, the 3D-meshes generated by the OBJAVERSE-trained models are high-quality and diverse, especially when compared to the generations from the ShapeNet-trained model. To quantify this observation, we asked crowdworkers to rate the diversity of *Bag* generations produced by the OBJAVERSE and ShapeNet trained models. When shown collections of nine randomly sampled generations from both models, workers rated the collection generated from the OBJAVERSE trained model as more diverse in appearance 91% of the time.

Our fruits and vegetables, composed of 9 varieties produces perhaps the highest quality output, a promising signal that can inspire future work in text-to-3D generation.

2.5.2 Instance Segmentation with CP3D

A key advantage of using simulated data for computer vision is that it is much cheaper to obtain expert annotations. Annotated OBJAVERSE objects can be rendered into images, allowing them to serve as a rich source of additional data that can be used to enhance model performance on 2D computer vision tasks. As a proof-of-concept demonstrating the effectiveness of this approach, we use segmented data from OBJAVERSE objects as auxiliary labels for training models on the LVIS dataset for Large Scale Instance Segmentation [87]. The LVIS dataset contains instance segmentation masks for 1200 object categories that occur throughout a set of 164k images. Recognition is especially challenging in this task due to the long tail of the object category distribution in this dataset. LVIS categories only contain an average 9



Figure 2.8: An existing ProcTHOR scene (left) and a semantically similar ProcTHOR generatable scene with OBJAVERSE objects (right).

instances across the dataset, so training on simulated data is a promising approach for overcoming the challenges of learning in this low-sample regime.

Using the LVIS-annotated subset of OBJAVERSE, we introduce CP3D: an enhancement to the simple, but effective, copy-and-paste technique of [84]. Figure 2.7 shows an example of the setup for CP3D. Here, we render different views of 3D objects and paste them on-top of existing LVIS images. We render 5 distinct views of each object and cache them for use throughout training. During training, an image is selected for the copy-paste augmentation with 0.5 probability, and once selected, 1-3 images of randomly chosen LVIS-annotated OBJAVERSE objects are pasted onto the selected training image. The segmentation masks of the selected objects are added to the training image’s annotation as well. Object images and masks are randomly scaled and translated before being pasted. We use this strategy to finetune the pretrained ResNet-50 Mask-RCNN [91, 92] of [5]. As shown in Tab.B.1, simply finetuning this model for 24 epochs yields performance gains across several metrics.

2.5.3 Open-Vocabulary ObjectNav

In this section, we introduce open-vocabulary ObjectNav, a new task propelled by the vast diversity of objects that appear in OBJAVERSE. Here, an agent is placed at a random starting location inside of a home and tasked to navigate to a target object provided from a text description (*e.g.* “*Raspberry Pi Pico*”). To facilitate this task, we procedurally generate 10K new homes in ProcTHOR [52] fully populated with objects from OBJAVERSE-LVIS. Until now, ObjectNav tasks have focused on training agents to navigate to 20 or so target objects provided their category label [48, 202, 52], and existing interactive embodied AI simulations, including ProcTHOR, only include around 2K total objects across around 100 object types [136, 52, 233]. In this work, we take a large step to massively scale the number of target objects used in ObjectNav (**20** \rightarrow **OpenVocab**), the number of objects available in simulation (**2K** \rightarrow **36K**), and the number of object types of the objects (**100** \rightarrow **1.1K**).

Method	AP	APr	APc	APf
RFS [87]	23.7	13.3	23.0	29.0
EQLv2 [235]	25.5	17.7	24.3	30.2
LOCE [66]	26.6	18.5	26.2	30.7
NorCal with RFS [178]	25.2	19.3	24.2	29.0
Seesaw [244]	26.4	19.5	26.1	29.7
GOL [5]	27.7	21.4	27.7	30.4
GOL + CP3D	28.3	21.8	28.3	31.1

Table 2.1: Comparison of our approach (GOL+CP3D) against SoTA Mask-RCNN ResNet-50 models on LVIS. We report results for APr, APc, and APf which measure AP for categories that are rare (appear in 1-10 images), common (appear in 11-100 images), and frequent (appear in >100 images), respectively

Object placement. To make the placement of objects in the houses more natural, we use the OBJAVERSE-LVIS subset and annotate placement constraints for each object category. Specifically, we annotate if objects of a given category typically appears on the floor, on-top of a surface, or on a wall. If instances of the object category may appear on the floor, we also annotate whether it may appear in the middle of the scene (*e.g.* a clutter object like a basketball) or on the edge of the scene (*e.g.* a toilet or a fridge). For objects placed on the floor, we also to automatically detect flat regions on top of the object’s mesh to place surface object types. The annotations are used by ProcTHOR for sampling objects to place in a scene. We also filter out OBJAVERSE-LVIS objects that do not appear inside of homes, such as a jet plane. Structural objects, like doors and windows, are inherited from ProcTHOR as they would require additional cleanup.

Object size correction. Objects in Sketchfab may be uploaded at unnatural scales (*e.g.* a plant being as large as a tower). We therefore scale the objects to be of a reasonable size for them to look natural in a house. Here, for each object category, we annotate the maximum bounding box dimension length that every instance of the object category should be scaled to. For example, we annotate the maximum bounding box dimension for bookcase to be 2 meters and fork to be 0.18 meters. If a 3D modeled bookcase then has a bounding box of $20\text{m} \times 6\text{m} \times 3\text{m}$, we shrink each side by a factor of $\max(20, 6, 3)/2 = 5$.

Preprocessing for AI2-THOR. We add support to AI2-THOR for loading objects on the fly at runtime. Previously, all objects had to be stored in a Unity build, but such an approach is impractical when working with orders of magnitude more object data. For each object, we compress it with Blender [44] by joining all of its meshes together, decimate the joined mesh such that it has at most 5K vertices, and bake all the UV texture maps into a single texture map. We then generate colliders using V-HACD [152] to support rigid-body interactions.

Approach. Given procedural houses populated with OBJAVERSE-LVIS, the task is to navigate to the proximity of a chosen target object and invoke a task-completion action when the target object is in sight, given an open-vocabulary description formed with the template “a {name} {category}”. The name is the object name given by its creator, which is often descriptive. We filter each by whether it is detected as being written in English by a language detector [109, 108], and fall back to a class-only description for non-English name. Examples of the possible expressions include “a victorian-monobike motorcycle”, “a unicorn pony”, or “a dino ghost lizard”. The agent, similar to the ones in [118], observes an RGB egocentric view of the environment, pre-processed by the visual branch of a frozen ResNet-50 CLIP model [195] – the target description is pre-processed by the corresponding text branch. We train the agent with DD-PPO [255] and evaluate on houses with floor plans, objects, and descriptions unseen in training. We use the AllenAct [252] framework to train our agent. Our trained agent achieves a success rate of 19.9%, for a random policy success of 5.1%. For more details about the experiment refer to the appendix.

2.5.4 Analyzing Robustness

A persistent bias present in many image datasets, *e.g.* ImageNet [210], is that the subjects of interest are generally photographed from a forward-facing, canonical, orientation. When, for example, taking a photograph of a television, few would choose to take this photograph crouched on the floor behind the television. This impact of this bias was studied by Alcorn *et al.* [4] who find that modern computer vision systems are highly susceptible to deviations from canonical poses. This is more than a theoretical problem: computer vision systems deployed in the real world will frequently encounter objects in non-canonical orientations and in many applications, *e.g.* autonomous driving, it will be safety critical that they behave well.

Given the above, we adopt the experimental design of Alcorn *et al.* and design, using OBJAVERSE assets, a benchmark for evaluating the robustness of state-of-the-art computer vision classification models to orientation shifts. In particular, for each object in our OBJAVERSE-LVIS subset, we render 12 images of the object from random orientations rendered upon a background with RGB values equalling the mean



Figure 2.9: Examples of objects rendered from random orientations and their 0-shot classification categories with the CLIP ViT-B/32.

Model	Random Rotation		Any Rotation		Δ Top-1
	Top-1	Top-5	Top-1	Top-5	
OpenAI-400M [195]					
RN50	21.4%	45.0%	43.9%	70.8%	22.5%
ViT-L/14	29.1%	54.5%	52.3%	77.2%	23.2%
LAION-400M [217]					
ViT-B/32	24.1%	48.5%	46.9%	74.2%	22.8%
ViT-L/14	30.6%	56.8%	50.5%	77.0%	19.9%
LAION-2B [216]					
ViT-B/32	27.0%	51.8%	50.3%	76.1%	23.3%
ViT-L/14	32.9%	59.2%	52.1%	78.0%	19.2%
ViT-H/14	32.3%	58.8%	50.1%	77.3%	17.8%

Table 2.2: Evaluating 0-shot CLIP classification models on our rotational robustness benchmark. Δ Top-1 denotes the difference between *Top-1 Any Rotation* and *Top-1 Random Rotation*. Models are strongly overfit to standard views of objects.

RGB values from ImageNet; see Fig. 2.9 for examples. This ability to, at scale, render objects from random viewpoints is a practical impossibility in the real world but is made trivial when using 3D assets. We then evaluate several modern open-domain image-classification networks (constrained to the $\approx 1,200$ LVIS categories) on these images and report 4 metrics for each model. These metrics include:

- *Top-1 Random Rotation* – the frequency with which a model correctly classifies an image as belonging to the respective LVIS category.

- *Top-1 Any Rotation* – the frequency with which a model classifies an image correctly from at least one of the 12 random orientations.

This second metric is diagnostic and serves to represent a model’s performance when shown an object from a canonical pose. We also have *Top-5* variants of the above metric where the correct category need only be in the top 5 predictions from the model. We report our results in Tab. 2.2 in which we evaluate a variety of performant pretrained models. Comparing the gap in performance between the *Top-k Random Rotation* and *Top-k Any Rotation* metrics we find that model performance dramatically degrades when viewing objects from unusual orientations.

2.6 Conclusion

We present OBJAVERSE, a next-generation 3D asset library containing 818K high-quality, diverse, 3D models with paired text descriptions, titles, and tags. As a small glimpse of the potential uses of OBJAVERSE, we present four experimental studies showing how OBJAVERSE can be used to power (1) generative 3D models with clear future applications to text-to-3D generation, (2) improvements to classical computer vision tasks such as instance segmentation, (3) the creation of novel embodied AI tasks like Open Vocabulary Object Navigation, and (4) quantifying the rotational robustness of vision models on renderings of objects. We hope to see OBJAVERSE enable a new universe of new applications for computer vision.

Chapter 3

Phone2Proc

3.1 Abstract

Training embodied agents in simulation has become mainstream for the embodied AI community. However, these agents often struggle when deployed in the physical world due to their inability to generalize to real-world environments. In this paper, we present Phone2Proc, a method that uses a 10-minute phone scan and conditional procedural generation to create a distribution of training scenes that are semantically similar to the target environment. The generated scenes are conditioned on the wall layout and arrangement of large objects from the scan, while also sampling lighting, clutter, surface textures, and instances of smaller objects with randomized placement and materials. Leveraging just a simple RGB camera, training with Phone2Proc shows massive improvements from 34.7% to 70.7% success rate in sim-to-real ObjectNav performance across a test suite of over 200 trials in diverse real-world environments, including homes, offices, and RoboTHOR. Furthermore, Phone2Proc’s diverse distribution of generated scenes makes agents remarkably robust to changes in the real world, such as human movement, object rearrangement, lighting changes, or clutter.

3.2 Introduction

The embodied AI research community has increasingly relied on visual simulators [126, 215, 265] to train embodied agents, with the expectation that the resulting policies can be transferred onto robots in the physical world. While agents trained within simulated environments have shown increased capabilities, progress in successfully deploying these policies onto physical robots has been limited.

Robots trained in simulation must overcome daunting challenges if they are to work effectively in a real space such as our home. First, they must overcome the generalization gap between the limited set of simulated environments they are trained on and the test scene of interest. In practice, policies trained to perform complex visual tasks with reinforcement learning struggle to perform well in novel scenes with novel layouts and object instances. Second, they must work in realistic environments where we live and work, which are often full of clutter, with objects that keep being moved around, with people in and out of the scene and with lighting changes. In short, we expect our agents to learn from a small set of training data points and generalize not just to a single test data point, but to a distribution of test data that is often semantically distant from the training data. Today’s methods are a ways away from delivering such performant, robust, and resilient robots [49, 31].

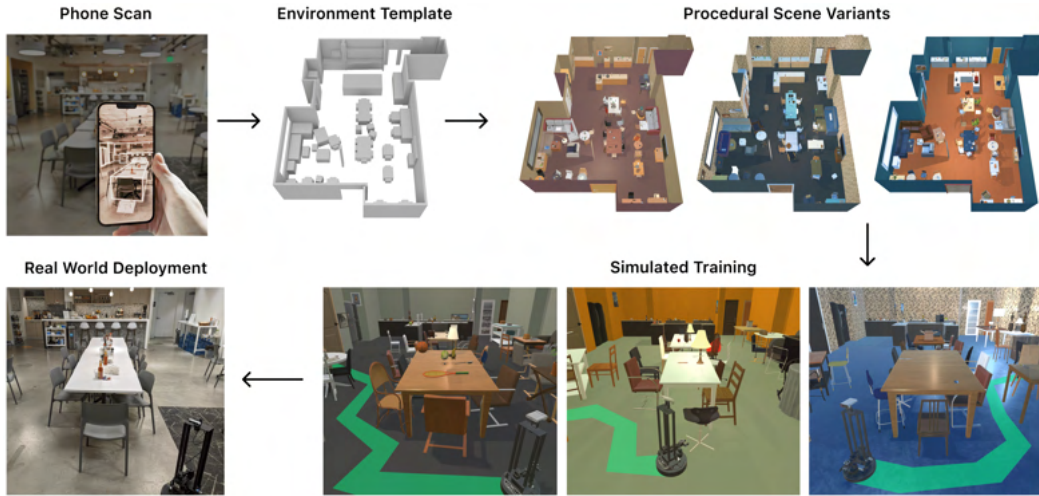


Figure 3.1: Successfully deploying agents trained in simulation to the real world has generally proved fraught - we present PHONE2PROC, a simple approach that uses a cellphone to scan an environment and procedurally generate targeted training scene variations of that location, whose usage results in successful and robust agents in the real environment.

In this work, we present PHONE2PROC, which represents a significant advancement towards the goal of creating performant, robust, and resilient robots. Instead of training policies in simulated environments that may be semantically distant from the target physical scene, PHONE2PROC efficiently generates a distribution of training environments that are semantically similar to the target environment. This significantly reduces the generalization gap between the training and target distributions, resulting in more capable robots.

PHONE2PROC utilizes a freely available mobile application to quickly scan a target environment and create a template of the surroundings, including the scene layout and 3D placements of large furniture. This template is then used to conditionally generate a fully interactive simulated world using ProcTHOR [52], closely mirroring the real-world space. Importantly, this single simulated environment is then transformed into a distribution of simulated worlds by randomizing objects, their placements, materials, textures, scene lighting, and clutter. This allows for the creation of arbitrary large training datasets that are semantically similar to the desired real-world scene.

We produce policies for object goal navigation using PHONE2PROC and deploy them onto a LoCoBot robot in the physical world. We conduct extensive evaluations with 234 episodes in five diverse physical environments: a 3-room and 6-room apartment, a test scene from RoboTHOR-real, a conference room, and a cafeteria. This represents one of the largest and most diverse studies of sim-to-real indoor navigation agents to date. Across all environments, PHONE2PROC significantly outperforms the state-of-the-art embodied AI model built with ProcTHOR, with an average improvement in success rate from 34.7% to 70.7%. Our robot is able to explore the scene efficiently and effectively navigate to objects of interest, even in the presence of clutter, lighting changes, shifts in furniture, and human movement. These strong navigation results are achieved using an **RGB-only camera**, **no depth** sensors, **no localization** sensors, and **no explicit mapping** components.

In summary, we present: (1) PHONE2PROC, a simple and highly effective method for reducing the generalization gap between datasets of simulated environments and a target environment in the real world, (2) large-scale real-world robotics experiments with 234 trials showing significant improvements for



Figure 3.2: **Examples of environment templates for our five target test environments.** These are produced by an iPhone scanning each environment using our iOS app that leverages Apple’s RoomPlan API. These environment templates contain the room layouts and some 3D locations for large furniture objects. They do not contain small objects, textures, lighting, etc.

PHONE2PROC compared to state-of-the-art models, and (3) experiments demonstrating the robustness of PHONE2PROC in the face of variations such as changes in lighting, clutter, and human presence.

3.3 Related Work

Navigation in Simulated Environments. Visual navigation [12, 8] is a popular task in the embodied AI community with benchmarks in several simulators [126, 265, 215, 198]. An effective approach is to use semantic mapping to explore environments efficiently [26, 27, 28, 82]. Kumar *et al.* [132] adapts mapping methods to condition the policy on the target. These works utilize agent pose and depth sensors to build their maps and localize the agent. In contrast, our method only relies on RGB information without any additional sensor. Other methods for navigation use scene priors [274], meta-learning [256], paired grid world environments [101], scene memory transformers [64], passive videos of roaming in a scene as a training cue [90] and expert human trajectories for imitation learning [203].

The community has also made progress in training embodied agents for navigation exclusively using RGB observations and barebones neural architectures. These include using frozen ImageNet trained visual encoders [283], learning visual encoders from scratch [49], and using CLIP [138] based encoders [118]. Gadre *et al.* [75] and [151] use an off-the-shelf exploration method and clip-based object localization to accomplish 0-shot object navigation. Deitke *et al.* [52] show the benefits of procedural generation for navigation and manipulation.

While the above works show promising results in simulation, most are not deployed and tested on real robots. We provide comparisons to the current state-of-the-art method, ProcTHOR [52], via large-scale real-world evaluations.

Sim-to-Real Transfer. While most models are evaluated only in simulation, for practical applications, policies learned in simulation must function in real life. Often, policies trained only in simulation can prove brittle or nonfunctional in transfer [104]. Chatopadhyay *et al.* [31] find that standard embodied agents significantly underperform (or fail) in the presence of realistic noise in the system. Truong *et al.* [242] compare the correlation of the performance of 4-legged robots navigating in simulation against the real world. They find that adding fidelity to the simulation does not help with the performance in the real world.

An alternate approach to higher fidelity is to add randomization to sensing or dynamics in simulation. This does help [240, 211], but too much randomization can degrade training efficacy [155], and hand-tuning appropriate randomization requires expert knowledge and does not scale. Some address this pitfall by leveraging real-world rollouts or inputs at train time to tune simulation randomization [57, 32]. However, these works are randomizing a subset of a well-parameterized dynamical system for a narrow task (swing-peg-in-hole or cabinet opening), as opposed to a more open-ended task or randomizing the entire visual appearance and object instances of the environment.

Recent works have deployed and measured policies on real robots [110, 14, 212, 58, 241] for the task of point goal navigation. In contrast to most, we use no mapping, explicit localization, or depth, as well as targeting a more complex task. We also test more extensively and in a wider variety of environments. Anderson *et al.* [9] study the sim-to-real gap for vision and language navigation and discover a crucial need for an oracle occupancy map and navigation graph. Our method does not require a manually annotated map and is robust to moving obstacles without the necessity for an additional dataset.

Real-to-Sim Transfer. Transferring observations from the real world to simulation can open up further capacities for training agents. This has been studied in the domain of object manipulation [144, 246]. [63] replicate an observed manipulation trajectory by predicting contact points and the forces. [107, 175] generate a 3D mesh of an object with articulation from observed interactions. [231, 10] infer simulation parameters for deformable object manipulation. [273, 243] learn cloth material recovery from videos. Our focus is on conditioning our procedural generation on the real scene rather than perfectly replicating the observation. [190] use a self-supervised technique to utilize unlabeled images for acquiring data for training scene graph models. We focus on generating 3D interactable environments for training embodied agents.

Navigation in Robotics. The robotics community has made progress in navigating robots with different embodiments in a diverse set of environments. Gupta *et al.* [89] combine a differentiable planner module with mapping to train a visual navigation agent end-to-end. In contrast to our work, their method assumes perfect odometry. Different methods have been used to build agents that follow demonstrated paths and trajectories or navigate [94, 131, 157, 156, 269, 213]. Shah *et al.* [221, 220] build models for open world navigation. The main focus of these works is on the low-level control systems of the robots, whereas our focus is on building end-to-end models for embodied visual reasoning.

3.4 Approach

We now present PHONE2PROC which generates a distribution of training environments that closely match the real world physical space we are interested in. We begin with a phone scan of a target scene (Sec 3.4.1), then condition on this scan to procedurally generate variations of the scene for training agents (Sec 3.4.2), and finally transfer onto a LoCobot robot that navigates in the physical world (Sec 3.4.3).

3.4.1 Scanning

PHONE2PROC is designed to optimize a robot’s performance within a desired real world environment. The first step in this process is to scan the target environment. This is accomplished using an iOS app that we built and will release using Apple’s freely available RoomPlan API [47]. Scanning a large apartment with several rooms only takes a few minutes, can be done using an iPhone or iPad and it outputs the environment template as a USDZ file.

The RoomPlan API provides us with a high-level bounding box template of the environment, which contains the room layouts and 3D placement of large objects that are visible to the camera. While scanning an environment, the app provides detailed real-time feedback about the construction of the scene to help the user capture a more accurate scan.

The resulting environment template includes the 3D locations and poses of walls, large objects, windows, and doors. Each object in the scan is assigned to one of 16 object types, including storage, sofa, table, chair, bed, refrigerator, oven, stove, dishwasher, washer or dryer, fireplace, sink, bathtub, toilet, stairs, and TV. Smaller objects, such as those typically on surfaces, are ignored. The metadata produced for each object includes the size, position, and rotation of its 3D bounding box, along with the forward-facing orientation. Doors and windows are provided as cutouts in the walls. Figure 3.2 presents examples of scanned environments showcasing the diversity of the layouts in our test set.

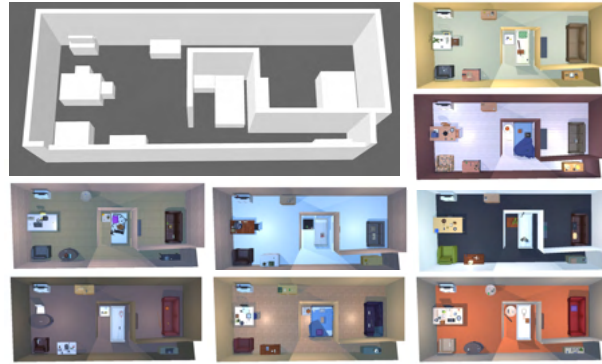


Figure 3.3: **Examples of Procedurally Generated Houses.** The procedural generation of the houses is conditioned on the target environment scanned using a phone. We are able to sample a rich and diverse set of scenes from this distribution with varying lighting, textures, objects and placements.

3.4.2 Environment-Conditioned Procedural Generation

Procedural generation of simulation environments allows for a vast diversity of scenes for agents to train on. Deitke *et al.* [52] begin with a high-level room specification (*e.g.* 2 bedroom house with a kitchen and living area) and create an environment that matches it. In contrast, our approach uses a scan of the target real-world environment to condition the generation and create variations of that scene. This process involves (a) parsing the environment template, (b) generating the scene layout, (c) sampling objects from the asset library to match scanned semantic categories, (d) accounting for object collisions in Unity, (e) populating the scene with small objects not captured by the scan, and (f) assigning materials and lighting elements.

PHONE2PROC parses the USDZ environment template produced by the iOS app and extracts wall, door and window positions and 3D large object bounding boxes. It then leverages ProcTHOR to generate a fully rendered scene in Unity and finally populates this scene using ProcTHOR’s asset database of 1,633 assets across 108 object types. The generation process is very fast and can generate 1000 procedural scene variants in around an hour with an 8 Quadro RTX 8000 GPU machine. We now provide further details on each of these steps.

Layout. The environment specification file contains the placement of walls in each room. Unlike walls generated in ProcTHOR-10K [49], which are only aligned to orthogonal axes, PHONE2PROC allows for a more diverse wall generation that can accommodate any scanned layout. Each wall’s specification comes from its 3D bounding box, width, height and a constant depth (*e.g.* 16cm for all walls) – which are used to produce a wall asset within the simulated environment. Placing walls produces the external boundary of the environment as well as its internal layout.

Rooms. We partition the space located within the external boundary walls into distinct rooms. A room is formed if the walls formed from the top-down 2D plane fully enclose a polygon. This is followed by floor and ceiling generation.

Windows and doors. The environment template specifies if each wall has cutouts for windows and doors. Our USDZ parser extracts the size and position of the holes along the wall. If the hole includes a cutout at the bottom of the floor, we place a door there; otherwise, we place a window. Here, we uniformly sample a door or window asset from the asset database and scale it appropriately.

For doors between connecting rooms, the sampled door may either include just a frame or both a frame with an openable door and a degree of openness sampled uniformly between 0.8 and 1.0. The room that the door opens into is randomly sampled. If the door is connected to the outside of the environment, we sample a door frame with an openable door component and fully close the door (to prevent agents from getting out of the scene).

Semantic objects. For each object in the template, we wish to sample an appropriate asset matching its semantic category. For each semantically similar ProcTHOR object candidate, we compute its 3D bounding box IoU with the object it may represent in the environment template and reject candidates with an IoU less than 75%. We then uniformly sample from the rest. The sampled asset’s position on the floor and its forward-facing direction come from the environment template’s corresponding object. We compute the vertical position of the object based on if it is on a surface (*e.g.* a couch on the floor or a television on top of a table) or attached to a wall (*e.g.* a wall television).

This procedure to find a matching asset can sometimes lead to large variations. For example, a table object may match a ProcTHOR coffee table, side table, or dining table and a TV may match a flat-screen TV or a vintage box television. Randomly sampling different asset instances in the library of a particular semantic object type makes agents more robust as they must learn to generalize to many visually distinct instances for each type.

Object collisions. We check to make sure that none of the placed objects collide with one another or the walls in the scene to avoid unrealistic configurations.

Small objects. After placing the large objects that match the scan, we generate smaller objects to be placed on top of them. Here, for instance, we might populate a bed with pillows or place fruits and plates on the counter. Unlike what happens in a 3D reconstruction, where all the objects are static, we are able to randomize the placement of small and target objects to produce many scene variations and prevent overfitting (See experiments in Sec ??).

Lighting. Scene lighting is randomized such that each room is guaranteed one light, and then additional lights are uniformly sampled throughout the scene scaled to the number of rooms. Each light is then randomized in its intensity, RGB values, shadow bias, and strength.

Materials. We randomize materials following ProcTHOR’s material randomization by sampling from sets of structure materials (*i.e.* wall, floor, and ceiling) and object materials.

Clutter. After all the semantic objects from the scan have been sampled and smaller objects have been sampled to be placed on top of them, we also sample additional clutter objects (such as boxes, dumbbells, pillows, and plants) that are placed on the floor of each room – similar to how a real house may have objects like kids toys thrown around. These objects prevent overfitting to particular paths in the environment and helps teach agents to avoid obstacles.

3.4.3 Transfer to Real World

We first detail our model architecture and training regime then discuss our robot and physical scenes.

Model and Training Details. Our goals are to: a) design models that do not depend on unrealistic sensory data in real indoor environments like agent or target localization, b) use only RGB observations since real world depth cameras offer few choices, generally come with small FOV and are fairly noisy, and c) create agents that are robust to clutter and changes in the environment.

PHONE2PROC provides a distribution of simulated worlds that are sampled to produce a large training set. These scenes differ in the placement of small objects, materials, lighting, clutter, etc. This allows us to train policies that do not overfit to a single scene configuration, but instead generalize to realistic scene variations.

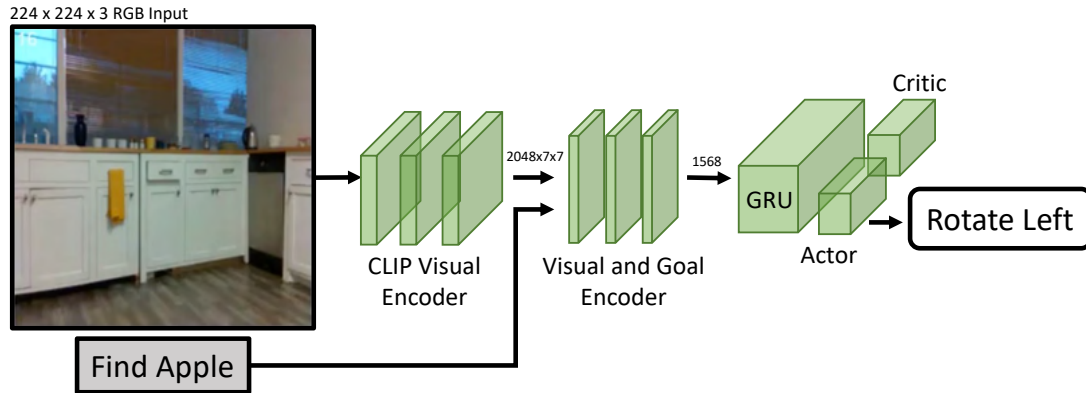


Figure 3.4: Our architecture is a simple GRU operating on the CLIP encoding of solely RGB input.

In terms of the model design, we adopt a simple architecture introduced in [118] and also used in [52]. The model uses a CLIP encoder to embed the visual observation (ego-centric RGB frame) followed by a GRU to capture temporal information (Fig 3.4). We pre-train our model on the ProcTHOR-10k dataset using the same training regime presented in [52] and then finetune on the Phone2Proc environments for the task of object navigation on 16 object categories. We use AllenAct [252] to train our models. More details on the training pipeline are provided in the appendix.

In principle, it is fairly straightforward to adopt a more complex model architecture, but we found this simple design to be highly performant not just in simulation but also in our real world experiments. Similarly, it is also easy to train agents for other tasks, including one involving object manipulation using an arm, since PHONE2PROC produces scenes that are fully interactive with support for all agents in AI2-THOR [126] including the arm-based agent [61].

During fine-tuning, we lower the learning rate to 0.00003 to avoid catastrophic forgetting of the skills learned in pre-training. We add a failed action penalty (0.05) in the reward shaping to encourage the agent to avoid hitting the obstacles in the environment. This is especially important as we deploy these models in the real world and would like to avoid damage to the environment or the robot. Instead of hand-tuning the camera parameters to match perfectly with the real world, the FOV of the camera in simulation is randomly sampled from a distribution approximating the real world.

Real-World Experiments. Models trained on procedurally-generated variants of the scene scans are then directly evaluated in real environments. We use 5 environments: a 3-room apartment, a large 6-room apartment, a real world test scene from RoboTHOR [49], a large re-configurable office conference room, and a cafeteria. Models are evaluated against 5 different target objects from 3 different starting locations in the environment. No training or calibration is performed in the real world.

No particular effort was made to arrange for ease of robotic experimentation. The goal was to use real environments in their most natural setting. The lighting, object instances, textures, and window views are not recognized by the PHONE2PROC scan and are thus unseen by the agent at training time. As there are many objects in the real-world scenes that are not present in ProcTHOR’s asset library (*e.g.* whiteboards, bicycle), there are several object categories in each environment that are novel to the agent. No additional information is used in the preparation/scanning step besides the output of the RoomPlan API.

Experiments are run on LoCoBot [88], a low-cost, platform about 60cm tall using the PyRobot API [168]. The agent’s discrete action space is look up/down, turn right/left (each 30°), move ahead 25cm, and a “done” action to indicate reaching the target. Actions are sampled using PyTorch’s categorical distribution.

FPS in real is ≈ 0.25 , and for practical reasons, physical trajectories were limited to 250 or 500 steps depending on the size of the environment.

3.5 Experiments

We provide extensive real-world evaluations of PHONE2PROC. In Sec. 3.5.1 we compare PHONE2PROC to PROCTOR in 5 diverse real environments. In Sec. 3.5.2 we show that PHONE2PROC performs as well as a privileged upper bound setting that utilizes a simulated counterpart of the real-world environment, painstakingly modeled by a digital artist. Sec. 3.5.3 illustrates the robustness of PHONE2PROC to various realistic changes in the environment, showing how PHONE2PROC hugely improves over using static reconstructions. Finally, we statistically analyze the significance of our results (Sec 3.5.4).

Scale of real-world evaluations. In aggregate we conduct 234 episodic evaluations in 5 diverse real-world environments. Our environments are large and challenging and each episode takes between 5 and 20 minutes to run. This represents one of the largest and most diverse real-world evaluation studies of sim-to-real indoor navigation agents. We put this number in the context of related works that provide 20 trials (1 scene) [26], 1 qualitative example [27], 36 trials (1 scene) [49] and 9 episodes (1 scene) [182]. A recent study for PointNav for studying Sim-vs-Real correlation [110] conducts 405 real trials but only uses a single laboratory setting.

Training models and baselines. All models use the same architecture and begin from a checkpoint trained on ProcTHOR-10k train set for the task of object navigation. This checkpoint is state of the art on 6 benchmark Embodied AI tasks [52]. This checkpoint is then fine-tuned for 5M steps on ProcTHOR-10K train with modifications detailed in section 3.4.3, and is henceforth referred to as PROCTOR or “baseline”. The PHONE2PROC models are environment-specific and are fine-tuned on 1K procedurally generated variants of scans of the relevant environment. Results for these are presented in Figure 3.5.

3.5.1 How Well Does Phone2Proc Work?

We evaluate PHONE2PROC in 5 diverse real-world environments: a 3-room apartment, a 6-room apartment, 1 RoboTHOR-Real apartment, a conference room, and a cafeteria. In each space, our model and the baseline are each evaluated for 15 trials (5 object categories with 3 agent initial locations per category). The 5 categories (apple, bed, sofa, television, and vase) were chosen to showcase both fixed objects whose locations can be learned (*e.g.* television) and small objects that must be searched for (*e.g.* apple). Where necessary (*e.g.* conference rooms do not usually contain beds), the bed and sofa are substituted for environment-appropriate objects such as chairs or garbage cans. Starting locations are geographically distributed and we avoid ones that would achieve trivial success.

Fig. 3.5 shows that in every real-world scene, PHONE2PROC performs remarkably well and significantly outperforms the PROCTOR baseline. In aggregate, PHONE2PROC achieves a Success Rate of 70.68% compared to 34.68% for PROCTOR. Overall, we find that the bigger environments with multiple rooms (RoboThor, 3 room apartment and 6 room apartment) are quite challenging for the baseline. PHONE2PROC on the other hand, performs very effectively in these scenes, that require it to perform long range exploration.

Table 3.1 compares PHONE2PROC with the same model architecture trained on Habitat [198] (implementation details on baseline training in the appendix). These results are presented on RoboTHOR-Real for 9 (instead of 15) episodes since Habitat only covers 3 of the 5 objects in our target set (bed, sofa, and television).

3.5.2 How Does Phone2Proc Compare To A

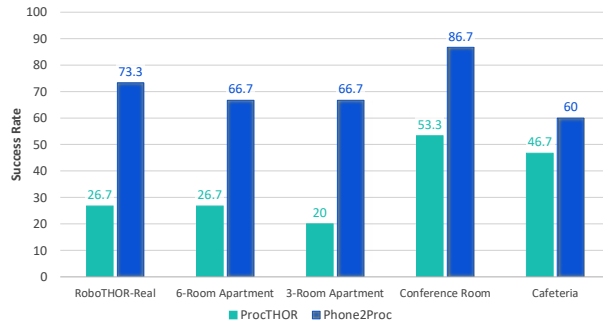


Figure 3.5: Results for PHONE2PROC vs ProcTHOR baseline in a variety of real environments. Each number represents fifteen trajectories - five objects from three starting locations.

Model	Success Rate	Episode Length
Habitat [198]	33.3	204.8
PROCTHOR [52]	33.3	92.5
PHONE2PROC (ours)	77.8	82.6

Table 3.1: Results of 9 trajectories evaluated in RoboTHOR-Real.

Privileged Upper Bound?

RoboTHOR test scenes come with a carefully and manually reconstructed simulation counterpart. This allows us to train a privileged model on this perfect replica. Producing this replica for a real scene took 5 days and is intractable practically but represents a theoretical upper bound, in terms of quality and correctness, for static 3D reconstruction methods that may employ sensors such as LiDAR. The model, referred to as *Reconstructed Simulation* or RECON, is only finetuned in this 1 scene. In simulation, RECON achieves 100% success since it overfits that scene easily.

We evaluate PHONE2PROC and RECON for 15 episodes in the RoboTHOR-real apartment (Fig. 3.6). PHONE2PROC is able to match the performance of this privileged baseline both in terms of Success and Episode length, which shows the effectiveness of our proposed approach to scan the target environment and train within its variations. In Sec 3.5.3, we show the pitfalls of this privileged model.


3.5.3 How Robust is Phone2Proc to Chaos?

In reality, our homes and offices aren't static and picture-perfect. Objects move around, furniture gets shifted, kids leave their toys on the floor, people keep moving around in the scene, lighting keeps changing throughout the day, and more! We evaluate the baseline model PROCTHOR, the privileged model RECON and our model PHONE2PROC in these settings (Fig. 3.7).

First, PROCTHOR does poorly in all settings, unsurprising given that it also fails on the episode with no variation. RECON performs well with no variations (consistent with Fig. 3.6). However, it performs very poorly when variations are introduced in the scene. When objects are moved around by just 1.5m, RECON fails, as it has memorized the location of every target object. Clutter and chair position adjustment confuses it, and the agent is simply unable to move around the scene and explore effectively. Moving the dining room furniture closer to the wall, as one might in a real house, produces interesting behavior. RECON calls the *Done* action for the television target when it sees the lamp. This is because in the original scan, the lamp was next to the television, and this is what the model likely memorized. RECON also fails

Model	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10	Task 11	Task 12	Task 13	Task 14	Task 15	Mean Len	Mean Success
Reconstructed Simulation (Upper bound)	250	239	19	92	217	250	58	76	89	34	250	7	67	51	250	86.3	73.3
Phone2Proc (Ours)	24	117	6	122	116	167	110	43	13	83	217	9	50	223	114	85.2	73.3

Figure 3.6: **Comparison with the reconstructed simulation.** We compare PHONE2PROC with the privileged reconstruction baseline among 15 different tasks (each task represents a different pair of agent’s initial location and target object). Each square shows the episode length of the corresponding model for each task. The red squares represent failed episodes and the blue ones indicate successful ones. Despite the baseline’s privileges, PHONE2PROC achieves a similar success rate and episode lengths.



Model	No Variance	Change in Lighting	Change in Target Location	Furniture Rearrangement	Clutter	People Moving Around	Change in Camera Parameters
ProctHOR	✗	✗	✗	✗	✗	✗	20.0%
Recon	✓	✓	✗	✗	✗	✗	26.7%
Phone2Proc	✓	✓	✓	✓	✓	✓	66.7%

Figure 3.7: Illustration of the scene disturbances used to comparatively evaluate models, along with their performances over each episode. The third column showcases the performance of models navigating to a vase when the object’s location is changed. For the last column, a full set of 15 trajectories is evaluated for each model. For other disturbances, we evaluated models for the target of Television.

when people move around during an episode. In stark contrast, PHONE2PROC is robust to every variation we tested, showing that procedurally generating variations of the scan helps train robust agents.

Finally, we tested all three models for robustness towards a change in camera parameters between simulation and the real robot. A full 15 trajectories were evaluated for each model trained with a wide vertical FOV and evaluated with a narrow one. PHONE2PROC is robust to this change, while RECON’s performance drops drastically.

3.5.4 Statistical Analysis

As described above, we have jointly evaluated PHONE2PROC and PROCTHOR models across 3 starting positions in 5 real environments with 5 target objects per environment (chosen from 7 unique types). Together this amounts to $(2 \text{ model types}) \times (3 \text{ positions}) \times (5 \text{ environments}) \times (5 \text{ targets}) = 150$ datapoints. In order to validate the statistical significance of our results, we follow a similar analysis as in [250] and model agent success using a logistic regression model in R [194]. In particular, here we model all exogenous variables as fixed effects and, as starting positions are inherently nested within environments, we include all environment and starting position interactions. When fitting this model, we obtain the coefficient estimates

	(Intercept)	Phone2Proc	Bed	Chair	GarbageCan	Sofa	TV	Vase
Coef.	0.17	0.33	0.00	0.41	-0.12	0.2	-0.03	0.13
p -value	0.31	<0.0001	0.97	0.02	0.59	0.13	0.78	0.27

where, for space, we have excluded coefficients corresponding to environments and locations, as none of these coefficients were statistically significant at a 0.05 level. As the above shows, the coefficient of interest (PHONE2PROC) is statistically significant even at a 0.0001 level. Interpreting these results, we see that when holding other factors constant, the use of a PHONE2PROC model is associated with $\exp(0.33) = 1.39$ times greater odds of success (95% confidence interval: $[1.2, 1.62]$) than when using a PROCTHOR model. Of all object categories, only the coefficient associated with chair was found to be statistically significant at a 0.05 level suggesting that chairs were associated with higher levels of success (*i.e.* may be generally easier to find). Altogether, we find strong statistical evidence suggesting that, across tested environments and object categories, PHONE2PROC was associated with higher success rates and that the esti



Figure 3.8: Qualitative results. These demonstrate the ability of PHONE2PROC models to navigate to their desired object. The top-down map is for visualization purposes only and is an approximation of the path taken by the agent.

3.5.5 Qualitative Analysis

Fig. 3.8 illustrates exemplary trajectories from each test environment with a few ego-centric RGB images that is the agent's only input. The trajectories show meaningfully different behavior for large vs. small objects (bed, sofa, and TV vs. apple and vase).

For large objects that don't change location drastically (*e.g.* row 1), the agent seems to initially localize itself using known landmarks that appear in the scan (similar object categories, for instance) and then demonstrate efficient motion towards the room which contains the target object. We observed during our trials that often, when an agent navigating toward a large target loses its way, it would double back to a familiar large object and then restart direct progress. Note that the agent has no ground truth localization and must rely on its observations and its recollection of environment layout and object presence.

Environment	Area (m ²)	Longest Path (m)	# Rooms	# Objects	# Scanned Objects
RoboTHOR-Real	34.5	8.1	4	51	14
6-Room Apartment	104.4	21.8	6	189	57
3-Room Apartment	65.4	8.2	3	105	26
Conference Room	98.3	10.0	1	48	32
Cafeteria	133.2	18.8	1	252	67

Table 3.2: The test real environments have a wide variety of layouts, usages, space, and object density. For visual layouts, see Fig. 3.2.

In contrast to big objects, for smaller items, the agent needs to explore efficiently to find the target. For instance, in the fourth row of Fig. 3.8, the agent searches for a small object that does not appear in the scan but is placed randomly in training rooms. Though again, it has no map, ground truth localization, or additional memory support, it demonstrates true exploration and high coverage of the possible area, ultimately achieving success (more qualitative results are provided in the supplementary videos).

3.5.6 Quantitative Analysis of The Environments

Results presented in Fig. 3.5 span a wide range of space usage, layout, and complexity as quantified in Table 3.2 to better demonstrate the power of PHONE2PROC. The conference room and cafeteria are large open spaces. The three living spaces require moving from room to room to locate objects. The 6-room apartment for instance, while most comparable in floor area to the conference room, is a long and narrow layout that requires hallway traversal for nearly every room transition. PHONE2PROC is most helpful in these environments over the baseline but makes a significant improvement in all layouts and semantic types of space.

3.6 Conclusion

In this paper, we introduced PHONE2PROC, a simple yet effective approach for training performant agents that are robust to the unpredictable nature of the real world. We demonstrated the capabilities of PHONE2PROC in five diverse environments and showed significant improvements in sim-to-real performance. Our environment-conditioned procedurally generated scenes are fully interactable, and we believe that future work will continue to explore its capabilities.

Bibliography

- [1] Panos Achlioptas, Judy Fan, Robert Hawkins, Noah Goodman, and Leonidas J Guibas. Shapeglot: Learning language for shape differentiation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8938–8947, 2019. 18
- [2] Adam, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents. *arXiv*, 2021. 80, 83
- [3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. *arXiv*, 2022. 5
- [4] Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4845–4854. Computer Vision Foundation / IEEE, 2019. 24
- [5] Konstantinos Panagiotis Alexandridis, Jiankang Deng, Anh Nguyen, and Shan Luo. Long-tailed instance segmentation using gumbel optimized loss. *arXiv preprint arXiv:2207.10936*, 2022. 23, 100
- [6] Allen Institute for AI. Rearrangement Challenge 2022. https://leaderboard.allenai.org/ithor_rearrangement_1phase_2022. 6, 97
- [7] Allen Institute for AI. RoboTHOR ObjectNav Challenge. <https://github.com/allenai/robothor-challenge>. 6
- [8] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On evaluation of embodied navigation agents. *arXiv*, 2018. 29, 92, 93
- [9] Peter Anderson, Ayush Shrivastava, Joanne Truong, Arjun Majumdar, Devi Parikh, Dhruv Batra, and Stefan Lee. Sim-to-real transfer for vision-and-language navigation. In *CoRL*, 2020. 30
- [10] Rika Antonova, Jingyun Yang, Priya Sundaesan, Dieter Fox, Fabio Ramos, and Jeannette Bohg. A bayesian treatment of real-to-sim for deformable object manipulation. *IEEE Robotics and Automation Letters*, 7:5819–5826, 2022. 30
- [11] Scott A Arvin and Donald H House. Modeling architectural design objectives in physically based space planning. *Automation in Construction*, 11(2):213–225, 2002. 79

- [12] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *ArXiv*, abs/2006.13171, 2020. 29
- [13] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv*, 2020. 92
- [14] Roberto Bigazzi, Federico Landi, Marcella Cornia, Silvia Cascianelli, Lorenzo Baraldi, and Rita Cucchiara. Out of the box: Embodied navigation in the real world. In *CAIP*, 2021. 30
- [15] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 15
- [16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020. 5, 80
- [17] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 7
- [18] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M Dollar. Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. *arXiv preprint arXiv:1502.03143*, 2015. 17, 18
- [19] Carnegie Mellon University. Locobot: an open source low cost robot. <http://www.locobot.org/>. 92, 100
- [20] Angel X Chang, Mihail Eric, Manolis Savva, and Christopher D Manning. Sceneseer: 3d scene design with natural language. *arXiv preprint arXiv:1703.00050*, 2017. 79
- [21] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv*, 2015. 7
- [22] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv*, 2015. 17, 18, 21, 80
- [23] Angel X. Chang, Will Monroe, Manolis Savva, Christopher Potts, and Christopher D. Manning. Text to 3d scene generation with rich lexical grounding. In *ACL*, 2015. 79
- [24] Angel X. Chang, Manolis Savva, and Christopher D. Manning. Learning spatial knowledge for text to 3d scene generation. In *EMNLP*, 2014. 7, 79, 80
- [25] Soravit Changpinyo, Piyush Kumar Sharma, Nan Ding, and Radu Soricut. Conceptual 12m: Pushing web-scale image-text pre-training to recognize long-tail visual concepts. In *CVPR*, 2021. 7
- [26] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Kumar Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. *Conference on Neural Information Processing Systems*, abs/2007.00643, 2020. 29, 34
- [27] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. *arXiv preprint arXiv:2004.05155*, 2020. 29, 34
- [28] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Kumar Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12872–12881, 2020. 29

- [29] Jorge L. Charco, Angel Domingo Sappa, Boris Xavier Vintimilla, and Henry O. Velesaca. Camera pose estimation in multi-view environments: From virtual scenarios to the real world. *Image Vis. Comput.*, 2021. 83
- [30] Aditya Chattopadhyay, Xi Zhang, David Paul Wipf, Rene Vidal, and Himanshu Arora. Structured graph variational autoencoders for indoor furniture layout generation. *arXiv preprint arXiv:2204.04867*, 2022. 7, 79
- [31] Prithvijit Chattopadhyay, Judy Hoffman, Roozbeh Mottaghi, and Aniruddha Kembhavi. Robustnav: Towards benchmarking robustness in embodied navigation. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15671–15680, 2021. 27, 29
- [32] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019. 29
- [33] Changan Chen, Ziad Al-Halah, and Kristen Grauman. Semantic audio-visual navigation. In *CVPR*, 2021. 83
- [34] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. Soundspaces: Audio-visual navigation in 3d environments. In *ECCV*, 2020. 83
- [35] Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. Text2shape: Generating shapes from natural language by learning joint embeddings. In *Asian conference on computer vision*, pages 100–116. Springer, 2018. 18
- [36] Xi Chen, Xiao Wang, Soravit Changpinyo, AJ Piergiovanni, Piotr Padlewski, Daniel Salz, Sebastian Goodman, Adam Grycner, Basil Mustafa, Lucas Beyer, et al. Pali: A jointly-scaled multilingual language-image model. *arXiv preprint arXiv:2209.06794*, 2022. 18
- [37] Rohan Chitnis, Tom Silver, Joshua B. Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning. *arXiv*, 2021. 83
- [38] Jaemin Cho, Jie Lei, Hao Tan, and Mohit Bansal. Unifying vision-and-language tasks via text generation. In *International Conference on Machine Learning*, pages 1931–1942. PMLR, 2021. 18
- [39] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014. 93
- [40] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv preprint arXiv:1602.02481*, 2016. 18
- [41] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv*, 2014. 93, 105
- [42] Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F Yago Vicente, Thomas Dideriksen, Himanshu Arora, et al. Abo: Dataset and benchmarks for real-world 3d object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21126–21136, 2022. 17, 18
- [43] Jasmine Collins, Shubham Goel, Kenan Deng, Achleshwar Luthra, Leon Xu, Erhan Gundogdu, Xi Zhang, Tomas F Yago Vicente, Thomas Dideriksen, Himanshu Arora, Matthieu Guillaumin, and Jitendra Malik. Abo: Dataset and benchmarks for real-world 3d object understanding. In *CVPR*, 2022. 80
- [44] Blender Online Community. Blender - a 3d modelling and rendering package. <http://www.blender.org>, 2018. 24
- [45] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013. 18
- [46] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017. 18

- [47] Afshin Dehghan, Yikang Liao, Hongyu Xu, Fengfu Li, Yang Yang, Alex Ke, Max Maung, Kota Hara, Peter Fu, Louis Gong, Yihao Qian, Zhuoyuan Chen, Guangyu Zhao, Tianxin Deng, Shaowei Liu, Chunyuan Cao, Rongqi Chen, Zhenlei Yan, Peiyin Heng, Jimmy Pan, Yinbo Li, Haiming Gang, Praveen Sharma, Antoine Tarault, Daniel Ulbricht, Haris BaigKai Kang, Joerg Liebelt, Peng Wu, and Feng Tang. 3d parametric room representation with roomplan, 2022. 30
- [48] Matt Deitke, Dhruv Batra, Yonatan Bisk, Tommaso Campari, Angel X Chang, Devendra Singh Chaplot, Changan Chen, Claudia Pérez D’Arpino, Kiana Ehsani, Ali Farhadi, et al. Retrospectives on the embodied ai workshop. *arXiv preprint arXiv:2210.06849*, 2022. 23
- [49] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. Robothor: An open simulation-to-real embodied ai platform. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3164–3174, 2020. 27, 29, 31, 33, 34
- [50] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, Luca Weihs, Mark Yatskar, and Ali Farhadi. Robothor: An open simulation-to-real embodied ai platform. In *CVPR*, 2020. 5, 6, 11, 12, 13, 83, 92
- [51] Matt Deitke, Aniruddha Kembhavi, and Luca Weihs. PRIOR: A Python Package for Seamless Data Distribution in AI Workflows, 2022. 83
- [52] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, et al. Proctor: Large-scale embodied ai using procedural generation. *Conference on Neural Information Processing Systems*, 2022. 17, 23, 28, 29, 31, 33, 34, 35, 100, 101, 105
- [53] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 7
- [54] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 17
- [55] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019. 80
- [56] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *ICRA*, 2022. 17, 18, 80
- [57] Yuqing Du, Olivia Watkins, Trevor Darrell, Pieter Abbeel, and Deepak Pathak. Auto-tuned sim-to-real transfer. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1290–1296. IEEE, 2021. 29
- [58] Daniel Dugas, Olov Andersson, Roland Y. Siegwart, and Jen Jen Chung. Navdreams: Towards camera-only rl navigation among humans. *ArXiv*, abs/2203.12299, 2022. 30
- [59] Kiana Ehsani, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Object manipulation via visual target localization. *arXiv*, 2022. 83
- [60] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ManipulaTHOR: A Framework for Visual Object Manipulation. In *CVPR*, 2021. 6, 13, 14, 83, 96
- [61] Kiana Ehsani, Winson Han, Alvaro Herrasti, Eli VanderBilt, Luca Weihs, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Manipulathor: A framework for visual object manipulation. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4495–4504, 2021. 33
- [62] Kiana Ehsani, Roozbeh Mottaghi, and Ali Farhadi. Segan: Segmenting and generating the invisible. In *CVPR*, 2018. 83

- [63] Kiana Ehsani, Shubham Tulsiani, Saurabh Gupta, Ali Farhadi, and Abhinav Kumar Gupta. Use the force, luke! learning to predict physical forces by simulating effects. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 221–230, 2020. 30
- [64] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 538–547, 2019. 29
- [65] Christiane Fellbaum. Wordnet. In *Theory and applications of ontology: computer applications*, pages 231–243. Springer, 2010. 104
- [66] Chengjian Feng, Yujie Zhong, and Weilin Huang. Exploring classification equilibrium in long-tailed object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3417–3426, 2021. 23
- [67] Di Feng, Christian Haase-Schuetz, Lars Rosenbaum, Heinz Hertlein, Fabian Duffhaus, Claudius Gläser, Werner Wiesbeck, and Klaus C. J. Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Trans. on Intelligent Transportation Systems*, 2021. 83
- [68] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG)*, 31(6):1–11, 2012. 7, 79, 80
- [69] Matthew Fontaine and Stefanos Nikolaidis. A quality diversity approach to automatically generating human-robot interaction scenarios in shared autonomy. *arXiv preprint arXiv:2012.04283*, 2020. 80
- [70] Jonas Freiknecht and Wolfgang Effelsberg. Procedural generation of multistory buildings with interior. *IEEE Transactions on Games*, 12(3):323–336, 2019. 79
- [71] Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942, 2021. 7, 79
- [72] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *International Journal of Computer Vision*, 129(12):3313–3337, 2021. 17, 18
- [73] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *International Journal of Computer Vision*, 129(12):3313–3337, 2021. 80
- [74] Samir Yitzhak Gadre, Kiana Ehsani, Shuran Song, and Roozbeh Mottaghi. Continuous scene representations for embodied ai. In *CVPR*, 2022. 83
- [75] Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. Clip on wheels: Zero-shot object navigation as object localization and exploration. *ArXiv*, abs/2203.10421, 2022. 29
- [76] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020. 20
- [77] Chuang Gan, Jeremy Schwartz, Seth Alter, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, Megumi Sano, Kuno Kim, Elias Wang, Damian Mrowca, Michael Lingelbach, Aidan Curtis, Kevin T. Feigelis, Daniel Bear, Dan Gutfreund, David Cox, James J. DiCarlo, Josh H. McDermott, Joshua B. Tenenbaum, and Daniel L. K. Yamins. Threedworld: A platform for interactive multi-modal physical simulation. In *NeurIPS (dataset track)*, 2021. 5, 6, 7
- [78] Chuang Gan, Yiwei Zhang, Jiajun Wu, Boqing Gong, and Joshua B Tenenbaum. Look, listen, and act: Towards audio-visual embodied navigation. In *ICRA*, 2020. 83
- [79] Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel LK Yamins, James J DiCarlo, Josh McDermott, Antonio Torralba, et al. The threedworld

- transport challenge: A visually guided task-and-motion planning benchmark for physically realistic embodied ai. *arXiv*, 2021. 83
- [80] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022. 15, 16, 21
- [81] Timnit Gebru, Jamie H. Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé, and Kate Crawford. Datasheets for datasets. *Comm. of the ACM*, 2021. 84, 90
- [82] Georgios Georgakis, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, and Kostas Daniilidis. Learning to map for active semantic goal navigation. *ArXiv*, abs/2106.15648, 2022. 29
- [83] Tobias Germer and Martin Schwarz. Procedural arrangement of furniture for real-time walk-throughs. In *Computer Graphics Forum*, volume 28, pages 2068–2078. Wiley Online Library, 2009. 7, 79, 80
- [84] Golnaz Ghiasi, Yin Cui, Aravind Srinivas, Rui Qian, Tsung-Yi Lin, Ekin D Cubuk, Quoc V Le, and Barret Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2918–2928, 2021. 23
- [85] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. In *CVPR*, 2018. 83
- [86] Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanaprasgam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. Kubric: a scalable dataset generator. In *CVPR*, 2022. 83
- [87] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5356–5364, 2019. 18, 22, 23
- [88] Abhinav Gupta, Adithyavairavan Murali, Dhiraj Prakashchand Gandhi, and Lerrel Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. *Advances in neural information processing systems*, 31, 2018. 33
- [89] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *International Journal of Computer Vision*, 128:1311–1330, 2017. 30
- [90] Meera Hahn, Devendra Singh Chaplot, and Shubham Tulsiani. No rl, no simulation: Learning to navigate without navigating. In *NeurIPS*, 2021. 29
- [91] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 23, 100
- [92] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 23, 100, 105
- [93] Padraig Higgins, Ryan Barron, and Cynthia Matuszek. Head pose as a proxy for gaze in virtual reality. In *Workshop on Virtual, Augmented, and Mixed Reality for HRI*, 2022. 83
- [94] Noriaki Hirose, F. Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics and Automation Letters*, 4:3184–3191, 2019. 30
- [95] Md Shahadat Hossain and Abdus Salam. Text-to-3d scene generation using semantic parsing and spatial knowledge with rule based system. *International Journal of Computer Science Issues (IJCSI)*, 14(5):37–41, 2017. 79

- [96] Ruizhen Hu, Zeyu Huang, Yuhan Tang, Oliver Matias van Kaick, Hao Zhang, and Hui Huang. Graph2plan: Learning floorplan generation from layout graphs. *ACM Trans. on Graphics*, 2020. 7, 79
- [97] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv*, 2022. 83
- [98] Gabriel Ilharco, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, Vaishal Shankar, Hongseok Namkoong, John Miller, Hannaneh Hajishirzi, Ali Farhadi, and Ludwig Schmidt. OpenCLIP, July 2021. 18
- [99] Allan Jabri, Andrew Owens, and Alexei Efros. Space-time correspondence as a contrastive random walk. *Advances in neural information processing systems*, 33:19545–19560, 2020. 19
- [100] Ajay Jain, Ben Mildenhall, Jonathan T Barron, Pieter Abbeel, and Ben Poole. Zero-shot text-guided object generation with dream fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 867–876, 2022. 18
- [101] Unnat Jain, Iou-Jen Liu, Svetlana Lazebnik, Aniruddha Kembhavi, Luca Weihs, and Alexander G. Schwing. Gridtopix: Training embodied agents with minimal supervision. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 15121–15131. IEEE, 2021. 29
- [102] Unnat Jain, Luca Weihs, Eric Kolve, Ali Farhadi, Svetlana Lazebnik, Aniruddha Kembhavi, and Alexander Schwing. A cordial sync: Going beyond marginal policies for multi-agent embodied tasks. In *ECCV*, 2020. 83
- [103] Unnat Jain, Luca Weihs, Eric Kolve, Mohammad Rastegari, Svetlana Lazebnik, Ali Farhadi, Alexander G Schwing, and Aniruddha Kembhavi. Two body problem: Collaborative visual task completion. In *CVPR*, 2019. 83
- [104] Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer, 1995. 29
- [105] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020. 7
- [106] Chao Jia, Yinfei Yang, Ye Xia, Yi-Ting Chen, Zarana Parekh, Hieu Pham, Quoc Le, Yun-Hsuan Sung, Zhen Li, and Tom Duerig. Scaling up visual and vision-language representation learning with noisy text supervision. In *International Conference on Machine Learning*, pages 4904–4916. PMLR, 2021. 18
- [107] Zhenyu Jiang, Cheng-Chun Hsu, and Yuke Zhu. Ditto: Building digital twins of articulated objects from interaction. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5606–5616, 2022. 30
- [108] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, H erve J egou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016. 24
- [109] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016. 24
- [110] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, S. Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5:6670–6677, 2020. 30, 34
- [111] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2real predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 2020. 83, 92
- [112] Peter K an and Hannes Kaufmann. Automatic furniture arrangement using greedy cost minimization. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 491–498. IEEE, 2018. 79, 80

- [113] Yash Kant, Arun Ramachandran, Sriram Yenamandra, Igor Gilitschenski, Dhruv Batra, Andrew Szot, and Harsh Agrawal. Housekeep: Tidying virtual households using commonsense reasoning. *arXiv preprint arXiv:2205.10712*, 2022. 80
- [114] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-sim: Learning to generate synthetic datasets. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4550–4559, 2019. 80
- [115] Siddharth Karamcheti, Dorsa Sadigh, and Percy Liang. Learning adaptive language interfaces through decomposition. *arXiv*, 2020. 83
- [116] Alexander Kasper, Zhixing Xue, and Rüdiger Dillmann. The kit object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934, 2012. 17, 18
- [117] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings for embodied ai. In *CVPR*, 2021. 13, 14, 83, 92, 94, 97
- [118] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings for embodied ai. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14809–14818, 2022. 24, 29, 33, 100
- [119] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14829–14838, 2022. 105
- [120] Seung Wook Kim, Yuhao Zhou, Jonah Philion, Antonio Torralba, and Sanja Fidler. Learning to simulate dynamic environments with gamegan. In *CVPR*, 2020. 83
- [121] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. 93, 96, 97, 102
- [122] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9601–9611, 2019. 18
- [123] Jing Yu Koh, Harsh Agrawal, Dhruv Batra, Richard Tucker, Austin Waters, Honglak Lee, Yinfei Yang, Jason Baldrige, and Peter Anderson. Simple and effective synthesis of indoor 3d scenes. *arXiv*, 2022. 83
- [124] Jing Yu Koh, Honglak Lee, Yinfei Yang, Jason Baldrige, and Peter Anderson. Pathdreamer: A world model for indoor navigation. In *ICCV*, 2021. 83
- [125] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Kumar Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv e-prints*, pages arXiv-1712, 2017. 15, 17, 20
- [126] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Kumar Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *ArXiv*, abs/1712.05474, 2017. 27, 29, 33
- [127] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv*, 2017. 5, 6, 11, 12, 13
- [128] Klemen Kotar and Roozbeh Mottaghi. Interactron: Embodied adaptive object detection. In *CVPR*, 2022. 83
- [129] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *ECCV*, 2020. 83
- [130] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. In *RSS*, 2021. 83

- [131] Ashish Kumar, Saurabh Gupta, David F. Fouhey, Sergey Levine, and Jitendra Malik. Visual memory for robust path following. In *NeurIPS*, 2018. 30
- [132] Gulshan Kumar, N Sai Shankar, Himansu Didwania, Brojeshwar Bhowmick, and K Madhava Krishna. Gcexp: Goal-conditioned exploration for object goal navigation. In *2021 30th IEEE International Conference on Robot & Human Interactive Communication (RO-MAN)*, pages 123–130. IEEE, 2021. 29
- [133] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallocci, Alexander Kolesnikov, et al. The open images dataset v4. *International Journal of Computer Vision*, 128(7):1956–1981, 2020. 17
- [134] Alina Kuznetsova, Hassan Rom, Neil Gordon Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallocci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4. *IJCV*, 2020. 7
- [135] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, et al. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, 2000. 18
- [136] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, et al. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. *arXiv preprint arXiv:2108.03272*, 2021. 15, 20, 23
- [137] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. In *CoRL*, 2021. 7
- [138] Guohao Li, Feng He, and Zhifan Feng. A clip-enhanced method for video-language understanding. *ArXiv*, abs/2110.07137, 2021. 29
- [139] Manyi Li, Akshay Gadi Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, Daniel Cohen-Or, and Hao Zhang. Grains: Generative recursive autoencoders for indoor scenes. *ACM Trans. on Graphics*, 2019. 79
- [140] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. In *CoRL*, 2022. 83
- [141] Zhengqin Li, Ting Yu, Shen Sang, Sarah Wang, Mengcheng Song, Yuhan Liu, Yu-Ying Yeh, Rui Zhu, Nitesh B. Gundavarapu, Jia Shi, Sai Bi, Hong-Xing Yu, Zexiang Xu, Kalyan Sunkavalli, Milos Hasan, Ravi Ramamoorthi, and Manmohan Chandraker. Openrooms: An open framework for photorealistic indoor scene datasets. In *CVPR*, 2021. 7
- [142] Ltd LIFULL Co. Lifull home’s dataset, 2015. Informatics Research Data Repository, National Institute of Informatics. 7
- [143] Joseph J Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing ikea objects: Fine pose estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 2992–2999, 2013. 17, 18
- [144] Vincent Lim, Huang Huang, Lawrence Yunliang Chen, Jonathan Wang, Jeffrey Ichnowski, Daniel Seita, Michael Laskey, and Ken Goldberg. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. *2022 International Conference on Robotics and Automation (ICRA)*, pages 8282–8289, 2022. 30
- [145] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 17
- [146] Ricardo Lopes, Tim Tutenel, Ruben M Smelik, Klaas Jan De Kraker, and Rafael Bidarra. A constrained growth method for procedural floor plan generation. In *Game-ON*, 2010. 7, 8, 64, 79

- [147] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vibert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. *Advances in neural information processing systems*, 32, 2019. 15
- [148] Andrew Luo, Zhoutong Zhang, Jiajun Wu, and Joshua B Tenenbaum. End-to-end optimization of scene layout. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3754–3763, 2020. 79
- [149] Haokuan Luo, Albert Yue, Zhang-Wei Hong, and Pulkit Agrawal. Stubborn: A strong baseline for indoor object navigation. *arXiv*, 2022. 83
- [150] Rui Ma, Akshay Gadi Patil, Matthew Fisher, Manyi Li, Sören Pirk, Binh-Son Hua, Sai-Kit Yeung, Xin Tong, Leonidas Guibas, and Hao Zhang. Language-driven synthesis of 3d scenes from scene databases. *ACM Transactions on Graphics (TOG)*, 37(6):1–16, 2018. 79
- [151] Arjun Majumdar, Gunjan Aggarwal, Bhavika Devnani, Judy Hoffman, and Dhruv Batra. Zson: Zero-shot object-goal navigation using multimodal goal embeddings. *ArXiv*, abs/2206.12403, 2022. 29
- [152] Khaled Mamou, E Lengyel, and A Peters. Volumetric hierarchical approximate convex decomposition. In *Game Engine Gems 3*, pages 141–158. AK Peters, 2016. 24
- [153] Fernando Marson and Soraia Raupp Musse. Automatic real-time generation of floor plans based on squarified treemaps algorithm. *International Journal of Computer Games Technology*, 2010. 7, 62, 79
- [154] Cristina Mata, Nick Locascio, Mohammed Azeem Sheikh, Kenny Kihara, and Dan Fischetti. Standardsim: A synthetic dataset for retail environments. In *ICIAP*, 2022. 83
- [155] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743. PMLR, 2018. 29
- [156] Xiangyun Meng, Nathan D. Ratliff, Yu Xiang, and Dieter Fox. Scaling local control to large-scale topological navigation. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 672–678, 2020. 30
- [157] Xiangyun Meng, Yu Xiang, and Dieter Fox. Learning composable behavior embeddings for long-horizon visual navigation. *IEEE Robotics and Automation Letters*, 6:3128–3135, 2021. 30
- [158] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM transactions on graphics (TOG)*, 30(4):1–10, 2011. 79
- [159] Meta AI. Habitat ObjectNav Challenge 2022. <https://aihabitat.org/challenge/2022>. 6, 14
- [160] Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. Text2mesh: Text-driven neural stylization for meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13492–13502, 2022. 18
- [161] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 83
- [162] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995. 21
- [163] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. StructureNet: Hierarchical graph networks for 3d shape generation. *arXiv preprint arXiv:1908.00575*, 2019. 20
- [164] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 909–918, 2019. 19

- [165] 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. 80
- [166] D. Morrison, P. Corke, and J. Leitner. Egrad! an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. *IEEE Robotics and Automation Letters*, 5(3):4368–4375, 2020. 17, 18
- [167] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable Manipulation Skill Benchmark with Large-Scale Demonstrations. In *NeurIPS (dataset track)*, 2021. 7
- [168] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. Pyrobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*, 2019. 33
- [169] Mark Murnane, Padraig Higgins, Monali Saraf, Francis Ferraro, Cynthia Matuszek, and Don Engel. A simulator for human-robot interaction in virtual reality. In *VRW*, 2021. 83
- [170] Yashraj Narang, Kier Storey, Ireteayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, Ankur Handa, and Dieter Fox. Factory: Fast contact for robotic assembly. In *RSS*, 2022. 83
- [171] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.*, 21:181:1–181:50, 2020. 80
- [172] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *European Conference on Computer Vision*, pages 162–177. Springer, 2020. 79
- [173] Nelson Nauata, Sepidehsadat Hosseini, Kai-Hung Chang, Hang Chu, Chin-Yi Cheng, and Yasutaka Furukawa. House-gan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *CVPR*, 2021. 79
- [174] Tianwei Ni, Kiana Ehsani, Luca Weihs, and Jordi Salvador. Towards disturbance-free visual mobile manipulation. *arXiv*, 2021. 83, 93
- [175] Neil Nie, Samir Yitzhak Gadre, Kiana Ehsani, and Shuran Song. Structure from action: Learning interactions for articulated object 3d structure discovery. *ArXiv*, abs/2207.08997, 2022. 30
- [176] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas A. Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei M. Zhang. Solving rubik’s cube with a robot hand. *ArXiv*, abs/1910.07113, 2019. 80
- [177] Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *AAAI*, 2022. 83
- [178] Tai-Yu Pan, Cheng Zhang, Yandong Li, Hexiang Hu, Dong Xuan, Soravit Changpinyo, Boqing Gong, and Wei-Lun Chao. On model calibration for long-tailed object detection and instance segmentation. *Advances in Neural Information Processing Systems*, 34:2529–2542, 2021. 23
- [179] Wamiq Reyaz Para, Paul Guerrero, Niloy Mitra, and Peter Wonka. Cofs: Controllable furniture layout synthesis. *arXiv preprint arXiv:2205.14657*, 2022. 79
- [180] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven M Seitz. Photoshape: Photorealistic materials for large-scale shape collections. *arXiv preprint arXiv:1809.09761*, 2018. 17, 18
- [181] Keunhong Park, Utkarsh Sinha, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Steven M Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5865–5874, 2021. 19

- [182] Ruslan Partsey, Erik Wijmans, Naoki Yokoyama, Oles Dobosevych, Dhruv Batra, and Oleksandr Maksymets. Is mapping necessary for realistic pointgoal navigation? *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17211–17220, 2022. 34
- [183] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in Neural Information Processing Systems*, 34:12013–12026, 2021. 79
- [184] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 103
- [185] Claudia Pérez-D’Arpino, Can Liu, Patrick Goebel, Roberto Martín-Martín, and Silvio Savarese. Robot navigation in constrained pedestrian environments using reinforcement learning. In *ICRA*, 2021. 83
- [186] Aleksei Petrenko, Erik Wijmans, Brennan Shacklett, and Vladlen Koltun. Megaverse: Simulating embodied agents at one million experiences per second. In *ICML*, 2021. 7
- [187] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. 20
- [188] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA*, 2016. 7
- [189] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 18
- [190] Aayush Prakash, Shoubhik Debnath, Jean-Francois Lafleche, Eric Cameracci, Gavriel State, Stan Birchfield, and Marc Teva Law. Self-supervised real-to-sim scene generation. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16024–16034, 2021. 30
- [191] Sarah Pratt, Rosanne Liu, and Ali Farhadi. What does a platypus look like? generating customized prompts for zero-shot image classification. *arXiv preprint arXiv:2209.03320*, 2022. 104
- [192] Xavier Puig, Kevin Kyunghwan Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *CVPR*, 2018. 7
- [193] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021. 19
- [194] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2022. 36
- [195] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021. 15, 18, 20, 24, 25, 105
- [196] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021. 5, 92
- [197] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020. 80
- [198] Santhosh K. Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel Xuan Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *arXiv*, 2021. 5, 7, 10, 11, 12, 13, 29, 34, 35, 105
- [199] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 18

- [200] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, 2021. 5
- [201] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021. 18
- [202] Ram Ramrakhya, Eric Undersander, Dhruv Batra, and Abhishek Das. Habitat-web: Learning embodied object-search strategies from human demonstrations at scale. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5173–5183, 2022. 15, 23
- [203] Ram Ramrakhya, Eric Undersander, Dhruv Batra, and Abhishek Das. Habitat-web: Learning embodied object-search strategies from human demonstrations at scale. In *CVPR*, 2022. 29, 92
- [204] Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In *ICCV*, 2021. 80
- [205] Daniel Ritchie, Kai Wang, and Yu-An Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *CVPR*, 2019. 79
- [206] Alex Rodriguez and Alessandro Laio. Clustering by fast search and find of density peaks. *science*, 344(6191):1492–1496, 2014. 80
- [207] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 15, 18
- [208] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011. 97
- [209] Julian F Rosser, Gavin Smith, and Jeremy G Morley. Data-driven estimation of building interior plans. *International Journal of Geographical Information Science*, 31(8):1652–1674, 2017. 79
- [210] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 17, 24
- [211] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 29
- [212] Assem Sadek, Guillaume Bono, Boris Chidlovskii, and Christian Wolf. An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments. *2022 International Conference on Robotics and Automation (ICRA)*, pages 9425–9431, 2022. 30
- [213] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. *ArXiv*, abs/1803.00653, 2018. 30
- [214] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *ICCV*, 2019. 5, 6
- [215] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9338–9346, 2019. 27, 29
- [216] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *arXiv preprint arXiv:2210.08402*, 2022. 15, 18, 25
- [217] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114*, 2021. 15, 18, 25

- [218] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016. 94
- [219] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. 94
- [220] Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart, and Sergey Levine. Recon: Rapid exploration for open-world navigation with latent goal models. In *CoRL*, 2021. 30
- [221] Dhruv Shah, Ajay Kumar Sridhar, Arjun Bhorkar, Noriaki Hirose, and Sergey Levine. Gnm: A general navigation model to drive any robot. *ArXiv*, abs/2210.03370, 2022. 30
- [222] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2556–2565, 2018. 15, 18
- [223] Bokui Shen, Fei Xia, Chengshu Li, Roberto Mart’in-Mart’in, Linxi (Jim) Fan, Guanzhi Wang, S. Buch, Claudia. Pérez D’Arpino, Sanjana Srivastava, Lyne P. Tchapmi, Micael Edmond Tchapmi, Kent Vainio, Li Fei-Fei, and Silvio Savarese. igibson, a simulation environment for interactive tasks in large realistic scenes. In *IROS*, 2021. 5, 6, 7, 11
- [224] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *CVPR*, 2020. 83
- [225] Arjun Singh, James Sha, Karthik S. Narayan, Tudor Achim, and P. Abbeel. Bigbird: A large-scale 3d database of object instances. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 509–516, 2014. 17, 18
- [226] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1746–1754, 2017. 79
- [227] Krishna Srinivasan, Karthik Raman, Jiecao Chen, Michael Bendersky, and Marc Najork. Wit: Wikipedia-based image text dataset for multimodal multilingual machine learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2443–2449, 2021. 15, 18
- [228] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Mart’in-Mart’in, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, S. Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *CoRL*, 2021. 83
- [229] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 7
- [230] Xingyuan Sun, Jiajun Wu, Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Tianfan Xue, Joshua B Tenenbaum, and William T Freeman. Pix3d: Dataset and methods for single-image 3d shape modeling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2974–2983, 2018. 17, 18
- [231] Priya Sundareshan, Rika Antonova, and Jeannette Bohg. Diffcloud: Real-to-sim from point clouds with differentiable simulation and rendering of deformable objects. *ArXiv*, abs/2204.03139, 2022. 30
- [232] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 92
- [233] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel Xuan Chang, Zsolt

- Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra. Habitat 2.0: Training home assistants to rearrange their habitat. In *NeurIPS*, 2021. 6, 11, 15, 20, 23
- [234] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019. 18
- [235] Jingru Tan, Xin Lu, Gang Zhang, Changqing Yin, and Quanquan Li. Equalization loss v2: A new gradient balance approach for long-tailed object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1685–1694, 2021. 23
- [236] Matthew Tancik, Vincent Casser, Xinchun Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-nerf: Scalable large scene neural view synthesis. In *CVPR*, 2022. 20, 83
- [237] Guy Tevet, Sigal Raab, Brian Gordon, Yonatan Shafir, Amit H Bermano, and Daniel Cohen-Or. Human motion diffusion model. *arXiv preprint arXiv:2209.14916*, 2022. 19
- [238] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016. 17
- [239] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl S. Ni, Douglas N. Poland, Damian Borth, and Li-Jia Li. Yfcc100m: the new data in multimedia research. *Comm. of the ACM*, 2016. 7
- [240] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017. 29
- [241] Joanne Truong, S. Chernova, and Dhruv Batra. Bi-directional domain adaptation for sim2real transfer of embodied navigation agents. *IEEE Robotics and Automation Letters*, 6:2634–2641, 2021. 30
- [242] Joanne Truong, Max Rudolph, Naoki Yokoyama, S. Chernova, Dhruv Batra, and Akshara Rai. Rethinking sim2real: Lower fidelity simulation leads to higher sim2real transfer in navigation. *ArXiv*, abs/2207.10821, 2022. 29
- [243] Huamin Wang, James F. O’Brien, and Ravi Ramamoorthi. Data-driven elastic models for cloth: modeling and measurement. *ACM SIGGRAPH 2011 papers*, 2011. 30
- [244] Jiaqi Wang, Wenwei Zhang, Yuhang Zang, Yuhang Cao, Jiangmiao Pang, Tao Gong, Kai Chen, Ziwei Liu, Chen Change Loy, and Dahua Lin. Seesaw loss for long-tailed instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9695–9704, 2021. 23
- [245] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel X Chang, and Daniel Ritchie. Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics (TOG)*, 38(4):1–15, 2019. 79
- [246] Luobin Wang, Runlin Guo, Quan Ho Vuong, Yuzhe Qin, Hao Su, and Henrik I. Christensen. A real2sim2real method for robust object grasping with neural surface reconstruction. *ArXiv*, abs/2210.02685, 2022. 30
- [247] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, et al. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*, 2022. 18
- [248] Xinpeng Wang, Chandan Yeshwanth, and Matthias Nießner. Sceneformer: Indoor scene generation with transformers. In *3DV*, 2021. 79
- [249] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *CVPR*, 2021. 13, 80, 83, 97
- [250] Luca Weihs, Aniruddha Kembhavi, Kiana Ehsani, Sarah M. Pratt, Winson Han, Alvaro Her-rasti, Eric Kolve, Dustin Schwenk, Roozbeh Mottaghi, and Ali Farhadi. Learning generalizable

- visual representations via interactive gameplay. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. 36
- [251] Luca Weihs, Jordi Salvador, Klemen Kotar, Unnat Jain, Kuo-Hao Zeng, Roozbeh Mottaghi, and Aniruddha Kembhavi. AllenAct: A framework for embodied AI research. *arXiv*, 2020. 13
- [252] Luca Weihs, Jordi Salvador, Klemen Kotar, Unnat Jain, Kuo-Hao Zeng, Roozbeh Mottaghi, and Aniruddha Kembhavi. Allenact: A framework for embodied ai research. *arXiv preprint arXiv:2008.12760*, 2020. 24, 33
- [253] Tomer Weiss, Alan Litteneker, Noah Duncan, Masaki Nakada, Chenfanfu Jiang, Lap-Fai Yu, and Demetri Terzopoulos. Fast and scalable position-based layout synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 25(12):3231–3243, 2018. 79, 80
- [254] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2019. 83, 94
- [255] Erik Wijmans, Abhishek Kadian, Ari S. Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020. 24, 101
- [256] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6743–6752, 2019. 29
- [257] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In *CVPR*, 2019. 83
- [258] Qi Wu, Cheng-Ju Wu, Yixin Zhu, and Jungseock Joo. Communicative learning with natural gestures for embodied navigation agents with human-in-the-scene. In *IROS*, 2021. 83
- [259] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6772–6782, 2021. 18
- [260] Wenming Wu, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. Data-driven interior plan generation for residential buildings. *ACM Trans. on Graphics*, 2019. 7, 79
- [261] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv*, 2018. 6
- [262] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 18
- [263] Fei Xia, Chengshu Li, Roberto Mart’ın-Mart’ın, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation. In *ICRA*, 2021. 83
- [264] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *CVPR*, 2018. 12
- [265] F. Xia, Amir Roshan Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. 27, 29
- [266] 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 Guibas, and Hao Su. SAPIEN: A SimULATED Part-based Interactive ENvironment. In *CVPR*, 2020. 5, 6, 7, 19
- [267] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Bongsoo Choy, Hao Su, Roozbeh Mottaghi, Leonidas J. Guibas, and Silvio Savarese. Objectnet3d: A large scale database for 3d object recognition. In *ECCV*, 2016. 7

- [268] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *The European Conference on Computer Vision (ECCV)*, 2022. 20
- [269] Linhai Xie, A. Markham, and Niki Trigoni. Snapnav: Learning mapless visual navigation with sparse directional guidance and visual reference. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1682–1688, 2020. 30
- [270] Kun Xu, Kang Chen, Hongbo Fu, Wei-Lun Sun, and Shi-Min Hu. Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models. *ACM Transactions on Graphics (TOG)*, 32(4):1–15, 2013. 7, 80
- [271] Zhenjia Xu, Beichun Qi, Shubham Agrawal, and Shuran Song. Adagrasp: Learning an adaptive gripper-aware grasping policy. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4620–4626. IEEE, 2021. 19
- [272] Karmesh Yadav, Santhosh Kumar Ramakrishnan, John Turner, Aaron Gokaslan, Oleksandr Maksymets, Rishabh Jain, Ram Ramrakhya, Angel X Chang, Alexander Clegg, Manolis Savva, Eric Undersander, Devendra Singh Chaplot, and Dhruv Batra. Habitat challenge 2022. <https://aihabitat.org/challenge/2022/>, 2022. 105
- [273] Shan Yang, Junbang Liang, and Ming C. Lin. Learning-based cloth material recovery from video. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4393–4403, 2017. 30
- [274] Wei Yang, X. Wang, Ali Farhadi, Abhinav Kumar Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *ArXiv*, abs/1810.06543, 2019. 29
- [275] Samir Yitzhak Gadre, Mitchell Wortsman, Gabriel Ilharco, Ludwig Schmidt, and Shuran Song. Clip on wheels: Zero-shot object navigation as object localization and exploration. *arXiv*, 2022. 83
- [276] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011*, v. 30,(4), July 2011, article no. 86, 30(4), 2011. 7, 79, 80
- [277] Rowan Zellers, Jiasen Lu, Ximing Lu, Youngjae Yu, Yanpeng Zhao, Mohammadreza Salehi, Aditya Kusupati, Jack Hessel, Ali Farhadi, and Yejin Choi. Merlot reserve: Neural script knowledge through vision and language and sound. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16375–16387, 2022. 15, 19
- [278] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022. 80
- [279] Shao-Kui Zhang, Wei-Yu Xie, and Song-Hai Zhang. Geometry-based layout generation with hyper-relations among objects. *Graphical Models*, 116:101104, 2021. 7, 79
- [280] Zaiwei Zhang, Zhenpei Yang, Chongyang Ma, Linjie Luo, Alexander G. Huth, Etienne Vouga, and Qixing Huang. Deep generative modeling for scene synthesis via hybrid representations. *ACM Trans. on Graphics*, 2020. 79
- [281] Kaiyu Zheng, Rohan Chitnis, Yoonchang Sung, George Konidaris, and Stefanie Tellex. Towards Optimal Correlational Object Search. In *ICRA*, 2022. 83
- [282] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 83
- [283] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Kumar Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364, 2017. 29
- [284] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv*, 2020. 7

Appendix A

ProcTHOR Appendix

3-Room Houses



Figure A.1: Examples of 3-room houses generated in PROCTOR-10K.

4-Room Houses



Figure A.2: Examples of 4-room houses generated in PROCTOR-10K.

5-Room Houses

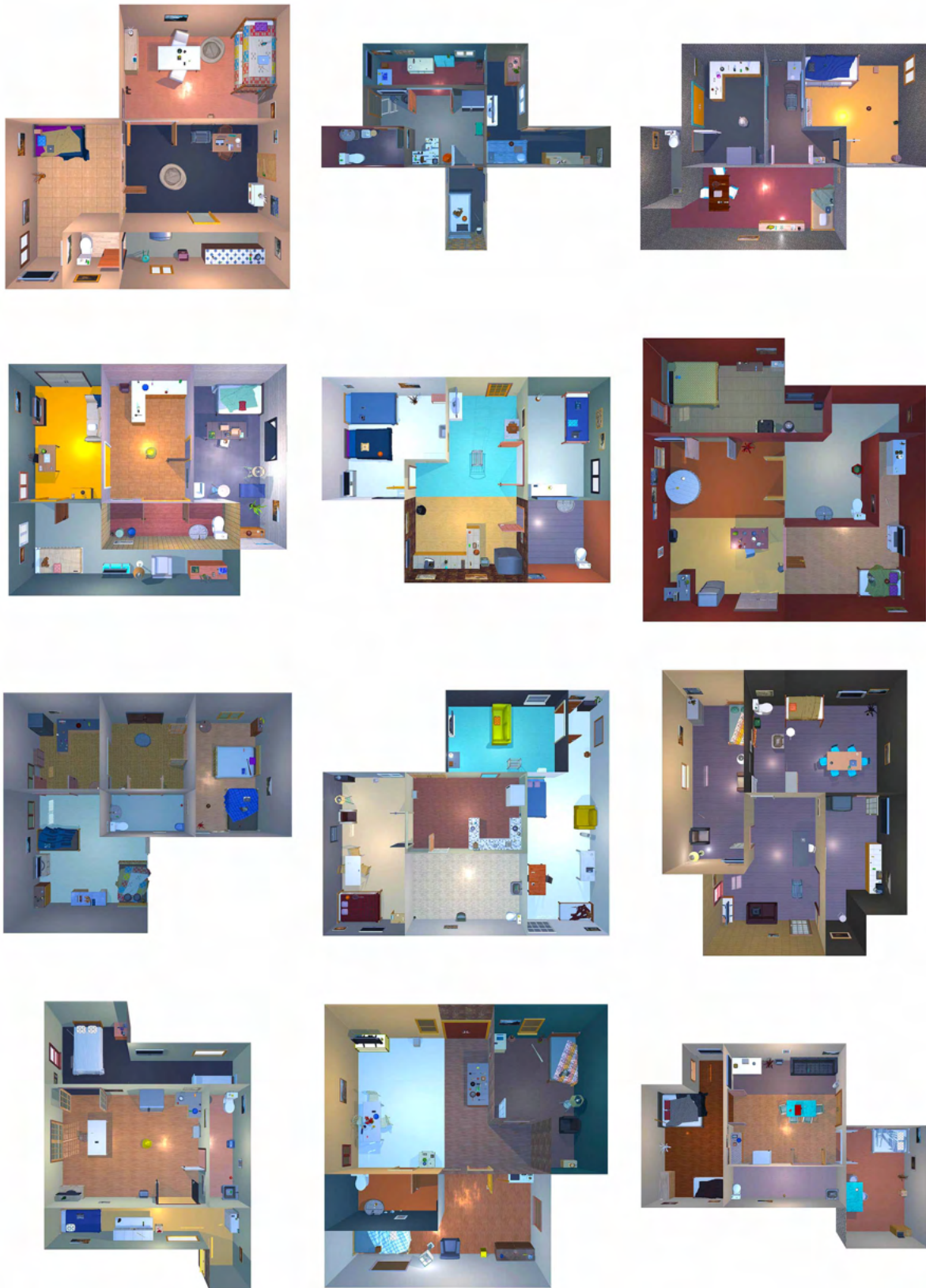


Figure A.3: Examples of 5-room houses generated in PROCTOR-10K.

6-Room Houses



Figure A.4: Examples of 6-room houses generated in PROCTOR-10K.

7+ Room Houses



Figure A.5: Examples of 7+ room houses generated in PROCTOR-10K.

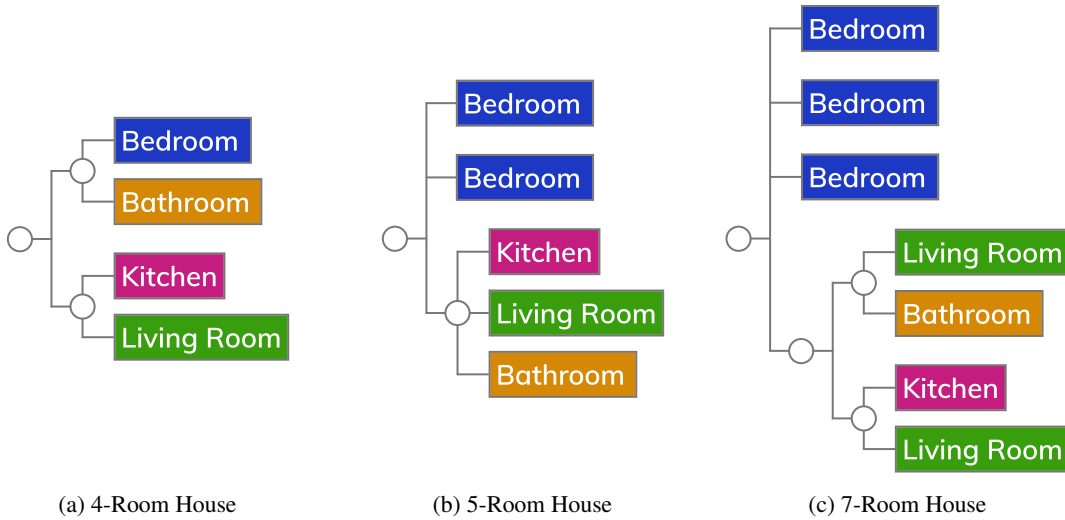


Figure A.6: Examples of room spec hierarchies used to sample differently sized houses.

A.0.1 Room Specs

Room specs provide the ability to specify the rooms that appear in a house, the relative size of each room, and how the rooms are connected with doors. Their idea was first proposed in [153]. A room spec is manually specified with a tree data structure.

Figure A.6a shows a simplified example of a room spec with four rooms: bedroom, bathroom, kitchen, and living room. In this room spec, there are two subtrees, comprising $\mathcal{Z}_{bb} = \{\text{bedroom, bathroom}\}$ and $\mathcal{Z}_{klv} = \{\text{kitchen, living room}\}$. At each level of the tree, there is a constraint that there must be a direct path connecting every child node of a parent. Thus, in our example, there will be a path between the bedroom and the bathroom, a path between the kitchen and the living room, and another path connecting \mathcal{Z}_{bb} to \mathcal{Z}_{klv} . We can also specify which room types we would prefer not to have a path between it and the parent. For example, we typically do not want the bathroom to have 2 doors, such as between it and the bedroom and between it and a room in \mathcal{Z}_{klv} .

Each tree node, below the root of the tree, is also assigned a growth weight, which approximates the relative size of the node compared to all other nodes that share the same parent. For instance, we might assign both \mathcal{Z}_{bb} and \mathcal{Z}_{klv} a growth rate of 1, to be roughly the same size. But, if we want the bedroom to take up roughly 60% of the \mathcal{Z}_{bb} 's area, then we might assign the bedroom a growth rate of 3 and the bathroom a growth rate of 2.

Room specs allow us to flexibly choose the distribution of houses we sample, allowing us to specify massive mansions, studio apartments, and anything in-between. Moreover, just a few room specs can go a long way. To generate our houses, we use 16 room specs, which each uses between 1 to 10 rooms. To generate the houses dataset, we assign a sampling weight to each of our room specs, and then use weighted sampling to sample a room spec for each house.

A.0.2 Sampling Floor Plans

The size and shape of the house are sampled to form the interior boundaries. Room specs specify the distribution over the dimensions of the house. Figure A.7 visualizes the process of sampling an

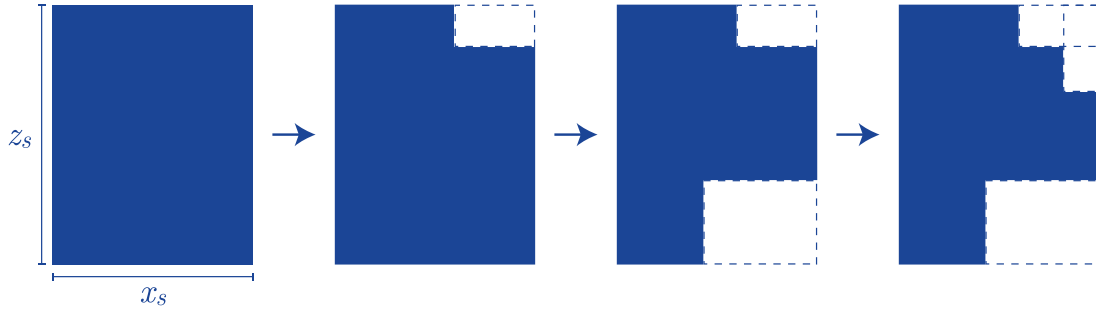


Figure A.7: An example of the interior boundary cut algorithm. The images show a top-down view of the house’s floor plan. First, we sample an interior boundary rectangle (x_s, z_s) , which is shown on the left. Then, we make n_c rectangular cuts to the corners of the rectangle to make the interior boundary of the house a more complex polygon. In this case, we make $n_c = 3$ cuts to form the final interior boundary, which is shown on the right.

interior boundary, where we first sample the size of the boundary and then make cuts to the corners to add randomness. The sampling starts off by choosing the initial upper bound of the top-down x and z size of the house, in meters, respectively denoted as x_s and z_s . Each dimension is an integer. In most room specs, each dimension is independently sampled from the discrete uniform distribution $x_s, z_s \sim U(\max(\ell_{\min}, \mu_a \sqrt{n_r} - \mu_a/2), \mu_a \sqrt{n_r} + \mu_a/2)$, inclusive. However, individual room specs can override the x_s and z_s sampling distributions. Here, n_r represents the number of rooms in the house, ℓ_{\min} is set to 2 and represents the minimum size of x_s and z_s , and μ_a is set to 3 and represents the average size of x_s and z_s per room.

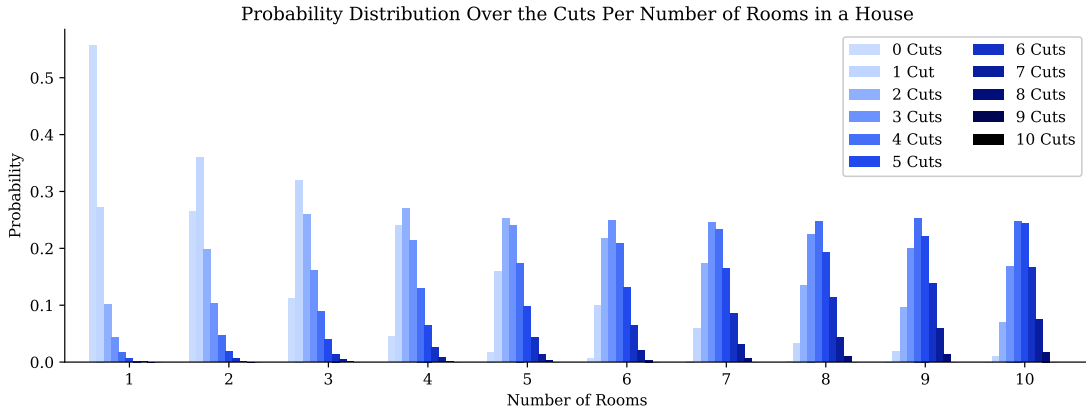


Figure A.8: The probability distribution over the number of cuts, n_c , made to the rectangular boundary (x_s, z_s) with respect to the number of rooms in the house, n_r . Notice that when there are more rooms in the house, the number of cuts in the distribution increases.

Once we have the rectangular boundary (x_s, z_s) , we then make several *cuts* to the outside of the rooms such that the interior boundaries can take on the shape of more complex polygons. The number of cuts, n_c , is sampled from the distribution $n_c \sim \lfloor 10 \cdot \text{Beta}(\alpha_c, \beta_c) + 1/2 \rfloor$, where $\alpha_c = n_r/2$ and $\beta_c = 6$. Figure A.8 shows the distribution that is formed with respect to the number of rooms in the house, n_r . When

there are more rooms, the probability distribution over the number of cuts increases. Since the range of the beta distribution is $(0, 1)$, the upper bound on the number of cuts is exactly 10.

The size of each cut is a rectangle, in meters, denoted by (c_x, c_z) . Both c_x and c_z are sampled from integer distributions. We sample from $c_x \sim U(1, \max(2, \min(x_s - 1, \lfloor a_{\max}/2 \rfloor) - 1))$, inclusive, where a_{\max} is set to 6 representing the maximum cut area. We then sample $c_z \sim U(1, a_{\max} - c_x)$. The position of where the cut happens is anchored to one of the 4 corners of the interior boundary, where the exact corner is independently and uniformly sampled each time.

Since the size of each cut is an integer, and the rectangular boundary sizes are also integers, we can efficiently represent the interior boundary with a (x_s, z_s) boolean matrix. Here, we could have 1s representing where the inside of the interior boundary and 0s representing the outside of the interior house boundary.



Figure A.9: An example of the recursive floor plan generation algorithm, given an interior boundary and the room spec in Figure A.6a. Here, we first divide the room into a “bedroom & bathroom” and a “kitchen & living room” zone. Then, within the “bedroom & bathroom” zone we place both the bedroom and bathroom, and within the “kitchen & living room” zone, we place both the kitchen and living room.

Given a room spec and an interior boundary, we use the algorithm proposed in [146] to divide the interior boundary into rooms. The algorithm recursively subdivides the interior boundary for each subtree in the room spec. Figure A.9 shows an example using Figure A.6a’s room spec. The algorithm first divides the interior boundary into two zones, the “bedroom & bathroom” zone and the “kitchen & living room” zone. The “bedroom & bathroom” zone then subdivides into two rooms, the bedroom and bathroom. Similarly, the “kitchen & living room” zone is also subdivided into two rooms, the kitchen and living room. The growth weight is used to approximate the size of each subdivision. By recursively subdividing the zones of each subtree, we satisfy the constraint that we can traverse between child nodes of the same parent in the room spec.

Finally, we scale the entire floor plan by $s \sim U(1.6, 2.2)$. Scaling the interior boundary to be larger provides more room for the agent to be able to navigate within the houses. Using a range of values also provides more variability on the size of the houses. We set the upper bound to 2.2 based on the empirical quality of the houses, where values above that often left too much empty space.

A.0.3 Connecting Rooms

Figure A.10 shows the 3 types of ways adjacent rooms may be connected. Specifically, rooms may be connected using 3 different types of connections: doorways, door frames, or open room connections. We determine which rooms should have doors between them based on the constraints in the room spec.



Figure A.10: An example of the 3 ways to connect different rooms, using either a doorway (left), door frame (middle), or open room connection (right).

Amongst adjacent rooms that may have doors between them, subject to the constraints in the room spec, we randomly sample which rooms have doors. We also impose the constraint that neighboring rooms in the room spec may have at most 1 room connected to it.

To choose the type of connection, we consider the rooms we are connecting. Specifically, we only allow open room connections and door frame connections between kitchen and living room room types. We impose this constraint because it would be unrealistic for a room like a bathroom to be fully visible from another room. For connecting room types that do support open room connections or door frames, we annotate the probability of sampling a doorway, door frame, and open room connection. Between a kitchen and living room the probability is 0.375 for sampling both an open room connection and a door frame connection, and 0.25 for sampling a doorway connection.

If a doorway or door frame is sampled, we filter to use a valid asset that is smaller than the wall connecting the rooms. For our generation, the minimum wall size is always greater than a single door size, but occasionally the filter might remove double doors from valid doors that can be sampled as they would be too big. The placement of the door is then uniformly sampled from anywhere along the wall. For doorways, the open direction is uniformly sampled. Finally, if the open state from any 2 doorways collides, we also use rejection sampling to potentially change the open direction and modify the placement of doorways.

Each house also has a permanently closed exterior door connecting to the outside. We prioritize placing this door in kitchen and living room room types, as it is unnatural to have to go through a bathroom or bedroom to go outside. However, in the case where the room spec does not include a kitchen or living room (*e.g.* if the room is a standalone bathroom), we randomly place a door to the outside in one of the remaining rooms.

A.0.4 Structure Materials

Wall materials. To choose the materials that make up the walls, we consider 2 families of wall materials: solid colors and texture-based materials. Our solid color materials consist of 40 unique colors of popular paint colors found in houses. We constrain ourselves to only using popular paint colors, so we do not randomize the walls to unrealistic colors such as bright green or yellow. For the texture-based materials, we annotate 122 different AI2-THOR materials to be suitable as wall materials. These include materials for brick textures, drywall textures, and tiling textures, amongst others.

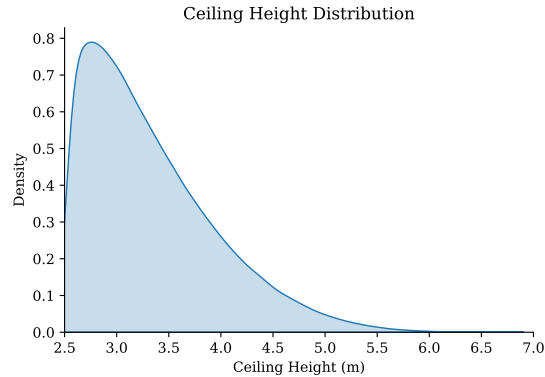


Figure A.11: The distribution of the ceiling height of each house, in meters.

Each wall in a room shares the same materials. For each room, we sample it if its materials are a solid color with $w_{solid} \sim \text{Bernoulli}(0.5)$. It is sometimes the case in real life that all rooms in a house share the same material (*e.g.* every room in an apartment is painted with white walls). We therefore also have a parameter $w_{same} \sim \text{Bernoulli}(0.35)$ that specifies if all rooms in the house will have the same material.

Ceiling material. The entire ceiling of the house is always assigned to a single wall material. If w_{same} , then the ceiling material is also set to the wall material. Otherwise, it is independently sampled with the same wall material sampling process.

Floor materials. We annotate 55 materials in AI2-THOR as floor materials. Most commonly, these materials are wood materials. For each room, we independently sample its floor material from the set of annotated floor materials. However, similar to wall materials, we independently sample $f_{same} \sim \text{Bernoulli}(0.15)$ that specifies if all rooms in the house will have the same material.

A.0.5 Ceiling Height

The ceiling height for the house, in meters, is sampled from $c_h \sim h_{\min} + (h_{\max} - h_{\min}) \cdot \text{Beta}(\alpha_h, \beta_h)$, where we set $h_{\min} = 2.5$, $h_{\max} = 7$, $\alpha_h = 1.25$, and $\beta_h = 5.5$. Figure A.11 shows the ceiling height distribution that is formed. All rooms in the house have the same ceiling height.

The minimum and mean values were chosen based on the typical height of an American apartment, while β_h allows some of the train houses to have much larger ceilings.

A.0.6 Lighting

Lighting Placement. Each procedural house places two types of lights: a directional light and point lights. The directional light is analogous to the sun in the scene, where only 1 is placed in each scene. Light from point lights are analogous to the light emitted from lightbulbs. We place a point light in each room near the ceiling, centered at the centroid of the room’s floor polygon. Using the centroid ensures that the light is always placed inside of the room, even for L-shaped rooms. Additionally, desk lamp and floor lamp objects have a point light associated with them.

Effects by the time of day. Skyboxes may appear at 3 different times of day: midday, golden hour, and blue hour. The time of day determines the intensity, hue, and direction of the ambient outdoor lighting. For each time of day, there exist multiple *skyboxes*, which dictate the lighting of the environment. Figure A.12 shows examples of how the time of day visually affects the scene. At this time, there are 16 midday



Figure A.12: Examples different skyboxes in a scene with a midday skybox (left), golden hour skybox (middle), and a blue hour skybox (right). Notice how the colors of the images differ and how the content outside of the window changes with the skybox.

skyboxes, 5 golden hour skyboxes, and 1 blue hour skybox, based on full 360-degree photos taken in Seattle and San Francisco.

A.0.7 Object Placement

In this section, we discuss how objects are placed realistically in the house. We hypothesize reasonable object placement is necessary in order to train efficient agents. For instance, if a toilet could appear anywhere in the house, the agent would have a much harder search problem, leading to longer episodes, than if the toilet was always in the bathroom. Moreover, we do not want objects to appear in unnatural positions, such as a fridge facing the wall, as it would make it unnatural, and even unusable, for interaction.

Finally, we do not always want objects to spawn independently. For instance, we might want a table to be surrounded by chairs. We achieve dependant sampling by developing SAGs, which are described in the section that follows.

Assets

The ProcTHOR asset database consists of 1,633 interactive household assets across 108 object types (see Appendix ?? for more details). The majority of assets come from AI2-THOR. Windows, doors, and counter tops are built into the exterior of rooms in AI2-THOR, which prevents us from spawning them in as standalone assets. Thus, we have also hand-built 21 windows, 20 doors, and 33 counter tops.

Asset Annotations. Our assets include several annotations that help us place them realistically in a house. Figure A.13 shows an example of the asset annotations used to place an arm chair. For an individual asset, we annotate its object type, computationally obtain its 3D bounding box, and partition assets of object types into training, validation, and testing splits. Then, we annotate how each object type might be spawned into the house. Annotating the 108 object types, as opposed to annotating the 1,633 individual assets, allows us to scale up the number of unique assets dramatically. Moreover, it does not require any new annotation to add an asset that can be grouped with an existing object type.

If instances of an object type cannot be placed independently on the floor, the rest of its annotations are not considered. For instance, we do not allow television object types to be placed alone on the floor, rather they are often placed on top of a television stand or mounted on the wall, which is discussed later in this section. Similarly, we also annotate small objects, like a fork, pen, and mug to not be placed

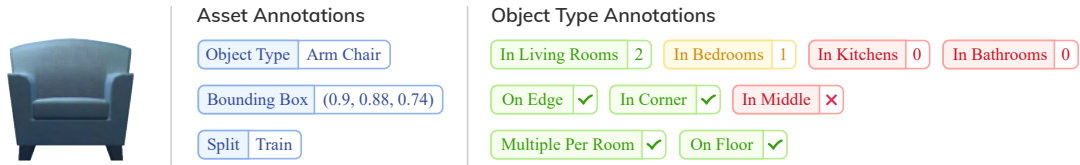


Figure A.13: An example of the asset annotations used to place an arm chair asset. This particular instance is annotated with its object type, bounding box, and split. Annotations about how it is placed in the house are done at an object type level, applying to all instances of that type.

independently on the floor. However, typical large object types, such as counter top, arm chair, or fridge object types can be placed independently on the floor.

Among the remaining object types, we annotate where and in which rooms the object type may appear. Each object type has a room weight, $r_w \in \{0, 1, 2, 3\}$, corresponding to how likely it is to appear in each room type. For each room type, a 0 indicates the object should never appear (e.g., a fridge in a bathroom); a 1 indicates the object may appear, but is unlikely; a 2 indicates that the object appears quite often; and a 3 indicates that the object nearly always appears (e.g., a bed in a bedroom). To determine where the object is placed, we annotate whether it may appear on the edge, in the corner, or in the middle of a room. For example, we annotate that a fridge can be placed on the edge or in the corner of the room, but not in the middle. We also annotate whether there can be multiple instances of an object type in a single room. Here, we annotate that multiple toilet object types cannot be in the same room, for instance.

Asset Splits. If an object type has over 5 unique assets, then those assets are partitioned into train, validation, and testing splits. Specifically, approximately $2/3$ of the assets are assigned to the train split, and approximately $1/6$ of the assets are assigned to each of the validation and testing splits. For object types that have 5 or fewer unique assets, they may appear in any split. In general, the more visual diversity an object type has, the more instances of that object type exist. For instance, there are many chair objects, but there are much fewer CD, toilet, and fork objects. Appendix ?? shows the precise count of each object type.

Semantic Asset Groups (SAGs)

A *Semantic Asset Group* (SAG) provides a flexible and diverse way to encode which objects may appear near each other. The power of SAGs comes in their ability to support randomized asset and rotational sampling. SAGs can be created and exported in seconds with our user-friendly drag-and-drop web interface.

Figure A.14 shows an example of how we might construct a SAG that has two chairs pushed into the side of a dining table. The SAG includes two chair samplers and a dining table sampler. Asset samplers contain a set of unique 3D modeled asset instances that may be sampled. When the SAG is instantiated, each asset sampler randomly chooses one of its instances. Asset samplers can also be linked, where multiple samplers sample the same asset instance each time. Here, linking may allow for multiple instances of the same chair to be placed at a dining table, instead of independently sampling a different chair for each sampler.

The ability to randomly sample assets to place in a SAG is incredibly expressive. For instance, consider a SAG with samplers for a TV stand, television, sofa, and arm chair. If each of these samplers can sample from just 30 different 3D modeled asset instances, then there are over 800k unique combinations of instances that can make be sampled from that SAG.

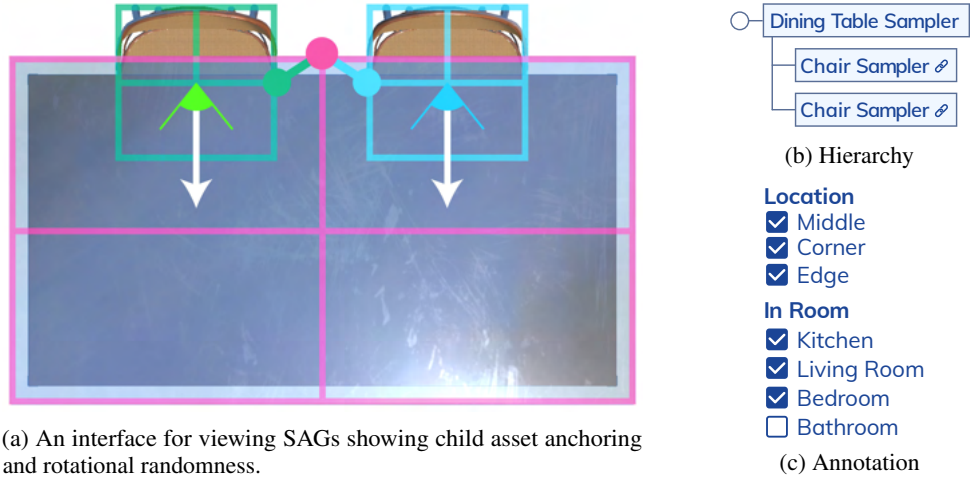


Figure A.14: An example of a semantic asset group (SAG), where two chair samplers are parented to a dining table sampler. Both chairs are anchored to the top middle of the table.

Asset samplers define how assets are positioned relative to one another. SAGs are constructed by looking at instances of asset samplers from their top-down orthographic images, such as the one shown in Figure A.14a. Here, both of the chair samplers are parented to the dining table sampler. Each child asset sampler is anchored to its parent asset sampler vertically in $\mathcal{V} = \{\text{TOP}, \text{CENTER}, \text{BOTTOM}\}$ and horizontally in $\mathcal{H} = \{\text{LEFT}, \text{CENTER}, \text{RIGHT}\}$. Each child asset sampler’s pivot position can similarly be set vertically in \mathcal{V} and horizontally in \mathcal{H} . For instance, in Figure A.14a, both chair samplers are anchored to the parent vertically on TOP and horizontally in the CENTER. But, the chair sampler on the left’s pivot position is vertically in the CENTER and horizontally on the RIGHT, whereas the chair sampler on the right’s pivot position is vertically in the CENTER and horizontally on the LEFT. Figure A.15 shows more examples of how a plant or floor lamp sampler may be positioned around an arm chair sampler. Each child asset sampler can then have an (x, y) offset, which is the distance from the parent sampler’s anchor point to the child sampler’s pivot position.



Figure A.15: Instantiations of a SAG that places a plant or floor lamp sampler \mathcal{S}_c around a parented arm chair sampler \mathcal{S}_p with anchor and pivot position annotations. Notice that the placement from \mathcal{S}_c reacts to the size of the asset sampled from \mathcal{S}_p . None of the examples have any offset.

The motivation for the relative positioning of asset samplers is to prevent the meshes from clipping into each other. For instance, with the same SAG in Figure A.14a, consider what would happen if the dining



Figure A.16: Rejection sampling is used to make sure objects placed in SAGs do not collide. *Left*: the chair collides with the dining table, and hence it is rejected; *Right*: none of the objects in the instantiated SAG collide with each other, so the SAG is accepted as valid.

table sampler samples a table that is double the size of the current table. Instead of the chairs being stuck in a fixed global position, and effectively colliding with the new dining table, the chairs will reactively move back, and be re-positioned to remain slightly tucked under the larger table. Moreover, consider that the size of instances that are sampled from an asset sampler are often quite different. For instance, one table might be square-ish, while another is elongated. If we only used a CENTER CENTER pivot and an offset, one would not be able to reliably place asset samplers, containing differently sized objects, directly beside each other without it resulting in clipping.

While setting anchoring and pivot positions solves many mesh clipping issues, some cases may still arise. Figure A.16 shows an example, where if our dining table sampler samples a short dining table, it may clip into certain chairs. Such issues are rare in practice, but object clipping would lead to less realistic and interactive houses. To solve the clipping issue, we use rejection sampling to resample the assets of a SAG until none of the 3D meshes of the sampled assets are clipping.

In PROCTOR-10K, we construct 18 SAGs, which can be instantiated with over 20 million unique combinations of assets. These include semantic asset groups for chairs around tables, pillows on top of beds, sofas and arm chairs looking at a television on top of a TV stand, faucets on top of sinks, and a desk with a chair, amongst others.

Floor Object Placement

We start object placement by first placing objects on the floor of the house. Objects are independently placed on a room-by-room basis, where we may first place objects in the bedroom and then place objects in the bathroom, without either affecting each other.

For each room, we filter the objects down into only using objects that have a room weight $r_w > 0$ in the given room type, and that have the annotation that they can be placed on the floor. Here, for instance, a chair object may have the annotation that it can be placed on the floor, but a knife object may not.

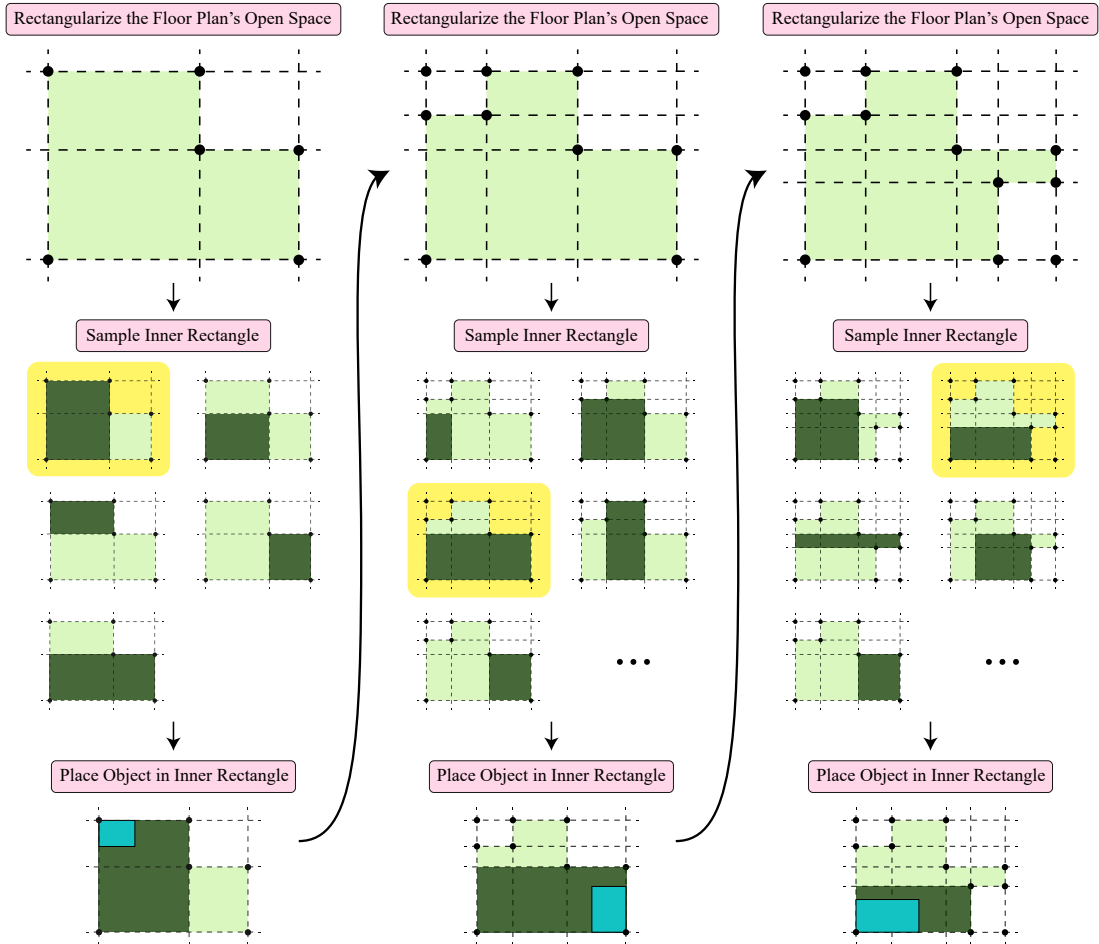


Figure A.17: Diagram detailing how floor objects are placed in a room. First, we rectangularize the top-down view of the room's open floor plan by drawing horizontal and vertical dividers from each corner point. Then, we construct all possible rectangles that are formed within the dividers. We then sample one of those rectangles and place the object within that rectangle. The sampled object's top-down bounding box (with margin) is shown in blue. The bounding box is then subtracted from the open floor plan before repeating the process again.

At this stage, we simplify rooms to just look at the top-down 2D bounding box that makes up the room in the floor plan. We also simplify objects to just look at its top-down 2D bounding box, of size (o_w, o_h) . These simplifications make it easier to determine if an object will fit in the room, specifically in a particular rectangle.

Figure A.17 illustrates the iterative process of placing objects in the scene. First, the polygon forming the area left to place an object is partitioned into rectangles. The rectangles come from drawing a horizontal and vertical grid line at all corner points of the open polygon. Here, we can easily obtain the largest rectangle remaining in the open room polygon. We sample $r_\ell \sim \text{Bernoulli}(0.8)$ to determine if the next object to be placed should be placed inside of the largest rectangle. Otherwise, we randomly choose amongst all possible rectangles, weighted by the area of each rectangle.

Once we have the rectangle (r_w, r_h) where the object should be placed in, we filter our objects to only those that would fit, both semantically and physically, in the rectangle. Semantically, we consider 3 scenarios: the rectangle being on the corner, edge, or middle of the room’s polygon.

If any of the rectangle’s corners is in a corner of the room, then we will place an object in that corner of the room. If multiple of the rectangle’s corners are in a corner of the room, then we uniformly sample a corner amongst one of those corners.

Now, we will filter down objects and asset groups to only consider:

1. Those that are annotated specifying that they can be placed in the corner of the room. For example, we might annotate a fridge to be placed in the corner of the room, but we might not annotate a SAG consisting of a dining table to be placed in the corner of the room.
2. The annotated split of the asset instance matches the current split of the generated house. See Appendix A.0.7 which talks about asset splits to create train/val/test homes.
3. The top-down bounding box of the object (with margin) must fit within the chosen rectangle. For a corner object, Figure A.18b shows the 2 valid rotations that this object may take on. Specifically, the back of the object may be against either wall. Then, we filter down remaining objects to only use those where the object’s bounding box fits within the rectangle’s bounding box; that is, $(o_h + w_{pad} \leq r_w \text{ and } o_w + w_{pad} \leq r_h)$ or $(o_h + w_{pad} \leq r_h \text{ and } o_w + w_{pad} \leq r_w)$. If both conditions are valid, we uniformly choose one of the rotations of the object’s bounding box.

We add margin around objects to make sure it is always possible to navigate around them. Objects to be placed in the middle of the room have $m_{pad} = 0.35$ meters of margin on each side. Objects on the edge or corner of the room have $w_{pad} = 0.5$ meters of margin only in front of the object, which enables objects to be placed directly beside it.

We sample an object or asset group that satisfies all of the previous conditions. If there are no objects or asset groups that satisfy all conditions, we continue to the next iteration and remove the selected rectangle from consideration. We slightly prioritize placing asset groups over standalone assets when possible. Once we have chosen an object or asset group, the bounding box with margin is then anchored to the corner of the rectangle, and hence to the corner of the room. We then subtract the object’s bounding box, with margin, from the open polygon representing the space remaining in the room before doing the same process again.

If the rectangle is along the edge, we sample $r_{edge} \sim \text{Bernoulli}(0.7)$ to determine if we should try to place an object on the edge of the rectangle, or if we should try and place it in the middle. If the rectangle is not along the edge or on the corner of the room, then we will always try to place an object in the middle of it. We use a similar filtering process, as the one described with edge rectangles, to filter down objects to those that only fit within the bounds of the rectangle. However, as depicted in Figure A.18a and Figure A.18c, edge objects can only have their backs to the wall, and middle objects can be rotated in any 90-degree rotation.

(a) Edge Rotations (b) Corner Rotations (c) Middle Rotations

Figure A.18: Valid rotations of objects when placed on the edge, corner, and middle of the room. Objects placed on the edge or corner of the room always have their backs to the wall. Objects in the middle of the scene can be rotated in any direction. By constraining rotations of objects, we ensure an object on the edge of the room, such as a fridge or drawer, can still be opened.

The iterative process of sampling a rectangle from the open polygon of the room, placing an object in that rectangle, and subtracting the bounding box formed by the object in the rectangle, continues on for r_i , where r_i is sampled from

$$r_i \sim \begin{cases} 1 & p = 1/200 \\ 4 & p = 2/200 \\ 5 & p = 4/200 \\ 6 & p = 20/200 \\ 7 & p = 173/200 \end{cases} . \quad (\text{A.1})$$

Sampling r_i allows us to infrequently have rooms in the house where there are very few objects, which is sometimes the case in real-world homes. It should also be noted that there can be more than r_i objects on the floor of the scene if some objects in the scene are in SAGs.

By iteratively choosing the largest, or near largest, rectangle in the room's open polygon, placing an object in it, and subtracting the object's bounding box with margin from the open room polygon, we enable great coverage across the entirety of the room, and hence the entirety of the house.

Wall Object Placement

After placing objects on floors, we then place objects on walls. We currently place window, painting, and television objects on the walls. Figure A.19 shows some examples. Window and television objects may appear in kitchen, living room, and bedroom room types. Paintings may appear in any room type.

Windows. Window objects are the first objects we place on the walls of the house. We only consider placing a window on walls that are connected to the outside of the house, such that we do not place a

Figure A.19: Examples of objects placed on the wall of a house, including a window (left), painting (middle), and television (right).

window between two indoor rooms. For each kitchen, living room, and bedroom in the house, we sample

$$n_w \sim \begin{cases} 0 & p = 0.125 \\ 1 & p = 0.375 \\ 2 & p = 0.5 \end{cases} \quad (\text{A.2})$$

maximum window objects to be placed.

For each wall in a given room, we look at the segment formed by each edge connecting 2 adjacent corners. If there is a floor object placed along that edge (or corner) of the wall, we subtract it from the segment. Here, the segment may break into different segments, where each segment is treated just like the original one. If the length of any segment is smaller than the minimum window size in the split, we remove the segment. We then use a uniform sample over the remaining segments, weighted by their lengths, to determine where to place the window. If no segments are longer than the smallest window, we move on to the next room in the house. A window smaller than the length of the segment is then uniformly placed somewhere along the sampled segment. The window is vertically centered along the wall between the floor and $w_{\max} = \min(3, c_h)$. All segments along the wall where the window was placed are removed from future sampling calls, and we continue this process n_w times.

Paintings. Painting objects are placed on the walls after window objects. They may be placed in any room. The maximum number of painting objects that are attempted to be placed in each room is sampled from

$$n_p \sim \begin{cases} 0 & p = 0.05 \\ 1 & p = 0.1 \\ 2 & p = 0.5 \\ 3 & p = 0.25 \\ 4 & p = 0.1 \end{cases} \quad (\text{A.3})$$

The placement of painting objects is similar to the placement of window objects. However, multiple painting objects may be placed along the same wall, so instead of removing the entire wall segment after an object is placed on it, we subtract the width of the painting from the segment. Moreover, we also allow painting objects to be placed above edge floor objects if the height of the edge object is less than 1.15 meters. Here, this allows for a painting to be above an object like a counter top, but not behind a taller object like a fridge.

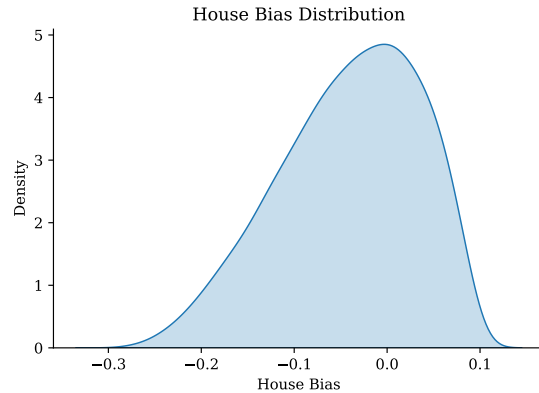


Figure A.20: The house bias distribution b_{house} that offsets the probability of attempting to spawn an object in a receptacle.

The vertical position of each painting is sampled at $o_y \sim w_{min} + (w_{max} - w_{min}) \cdot \text{Beta}(12, 12)$, where w_{min} is the maximum height of a floor object along the wall line. Here, we allow a painting to be placed above an object along the wall of the room, such as placing it above a counter top. Sampling from $\text{Beta}(12, 12)$ allows for some randomness in the sampling process while still having a large density near the center.

Televisions. Television wall objects may only be placed in living room, kitchen, and bedroom room types. Only 1 wall television may be placed in each room. From our annotations, television objects cannot be placed standalone on the floor. However, a television is often placed in a SAG, on top of an object like a TV stand. So as to not place too many television objects in the same room, we only filter by rooms that do not have a television object already in them. Amongst the remaining rooms, if the room type is a living room, we sample $\text{Bernoulli}(0.8)$ if we should try placing a wall television in the room. For kitchen and bedroom room types, we sample from $\text{Bernoulli}(0.25)$ and $\text{Bernoulli}(0.4)$, respectively. We only consider television objects that could be mounted to a wall (*i.e.* they do not have a base that is sticking out of the object). Television wall objects sample from the same vertical position distribution as painting objects, and follow the same placement on the walls as painting objects.

Surface Object Placement

After placing objects on the floor and wall of the house, we focus on placing objects on the surface of the floor objects just placed. For example, we may place objects like a coffee machine, plate, or knife on of a receptacle like a counter top.

For each receptacle object, we approximate the probability that each object type appears on its surface. We use the hand-modeled AI2-iTHOR or RoboTHOR rooms to obtain these approximations. Here, we compute the total number of times each object type is on the receptacle type and divide it by the total number of times the receptacle type appears across the scenes.

For each receptacle placed on the floor, we look at the probability of each object type p_{spawn} that it has been placed on that receptacle. We then iterate over the object types that may be on the receptacle. For each object type, we try spawning it on the receptacle if $\text{Bernoulli}(p_{spawn} + b_{house} + b_{receptacle} + b_{object})$, where

- b_{house} denotes the additional bias of how likely objects are to be spawned on receptacles in this particular house. Each house samples

$$b_{house} \sim (b_{house-max} - b_{house-min}) \cdot \text{Beta}(3.5, 1.9) + b_{house-min}, \quad (\text{A.4})$$

where $b_{house-min} = -0.3$ and $b_{house-max} = 0.1$. Figure A.20 shows the distribution that b_{house} forms. Using a house bias allows for some houses to be much cleaner or dirtier than others, whereas cleaner houses would have more objects put away that are not on receptacles.

- $b_{receptacle}$ denotes the additional bias of how likely an object is to be spawned on a receptacle. The default receptacle bias is 0.2, which is only overwritten by shelving unit (0.4 bias), counter top (0.2 bias), arm chair (0 bias), and chair (0 bias). Receptacle biases were manually set based on the empirical quality of the houses.
- b_{object} denotes the additional bias of how likely a particular object is to spawn in the scene. By default, b_{object} is set to 0, and overwritten by house plant (0.25 bias), basketball (0.2 bias), spray bottle (0.2 bias), pot (0.1 bias), pan (0.1 bias), bowl (0.05 bias), and baseball bat (0.1 bias). Object biases were also manually set based on the empirical quality of the houses to ensure more target objects appear in each of the procedurally generated houses.

Note that $p_{spawn} + b_{house} + b_{receptacle} + b_{object}$ may be greater than 1, in which case we will always try to spawn the object on the receptacle, or less than 0, where we will never try to spawn the object on the receptacle.

To attempt to spawn an object of a given type on a receptacle, we will sample an instance of that object type and randomly try $n_{pa} = 5$ poses of the object to try and fit the object instance on the receptacle. If the object instance fits and does not collide with another object, we keep it there. Otherwise, we try another pose of the object on the receptacle until we reach n_{pa} attempted poses. If none of the attempted poses work, we continue on to the next object type that may be on the receptacle.

If the first object of a given type is placed successfully on a receptacle, we attempt to place $n_{or} \sim \min(s_{max}, \text{Geom}(p_{spawn}) - 1) - 1$ more objects of that type given type on the receptacle. Here, s_{max} is set to 3, representing the maximum number of objects of a type that may be on a receptacle. We ignore the biases to not have too many objects of a given type on the same receptacle.

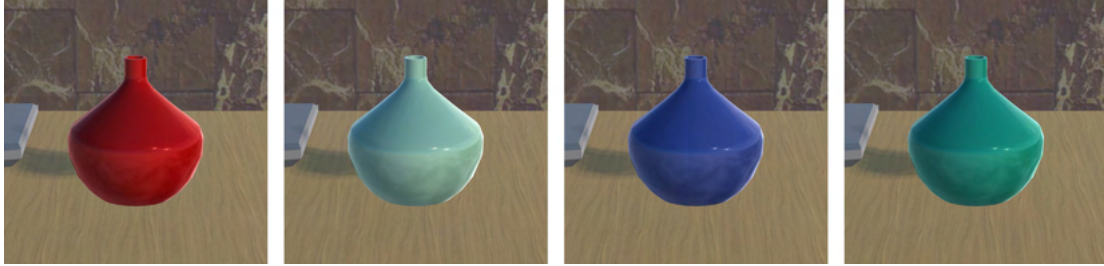
A.0.8 Material and Color Randomization

Several object types may have their color randomized to a randomly sampled RGB value. Specifically, for each vase, statue, or bottle in the scene, we independently sample from $r_c \sim \text{Bernoulli}(0.8)$ to determine if we should randomize the object’s color. These objects were chosen because they all still looked natural as any solid color. Figure A.21a shows some examples of randomizing the color of a vase.

For each training episode, we sample from $r_m \sim \text{Bernoulli}(0.8)$ to determine if we should randomize the default object materials in the scene. Wall, ceiling, and floor materials are left untouched to preserve w_{solid} and w_{same} sampling parameters. Materials are only randomized within semantically similar classes, which ensures objects still look and behave like the class they represent. For instance, an apple will not swap materials with an orange. Figure A.21b shows some examples of randomizing the materials in the scene.

A.0.9 Object States

We randomize object states to expose the agent to more diverse objects during training. For instance, instead of always having an open laptop or a clean bed, we randomize the openness of each laptop and if each bed is clean or dirty. Figure A.22 shows some examples. Our current set of state randomizations include:



(a) Examples of color randomization for a vase object. The original color is shown on the left. Notice that the vase still looks realistic with many possible colors.



(b) Examples of material randomization in ProcTHOR. Notice that only the objects randomize in materials, where the walls, floor, and ceiling remain the same.

Figure A.21: Examples of color randomization and material randomization in ProcTHOR.

- **Toggleing objects.** Floor lamp and desk lamp object types have their state toggled on or off.
- **Cleaning or dirtying objects.** Bed object types may appear as either clean or dirty.
- **Opening or closing objects.** Box and laptop object types may

toggling objects on or off (for floor lamp and desk lamp object types), setting objects to clean or dirty (for bed object types), and openness randomizations (for box and laptop object types).

A.0.10 Validator

Once a house is generated, we use a validator to make sure that the agent can successfully navigate to each room in the house, without modifying the scene through interaction (*e.g.* moving an object out of the way). Specifically, we first make sure the agent can teleport to a location inside the house. Then, from that position, we perform a BFS over neighboring positions on a 0.25×0.25 meter grid to obtain all reachable positions from the agent’s current position. The validator checks to make sure that every room in the house has at least 5 reachable positions on the grid. If the validator fails, we resample a new house using the same room spec, so as to not change the distribution of room specs that we sample from.

A.0.11 Related Works

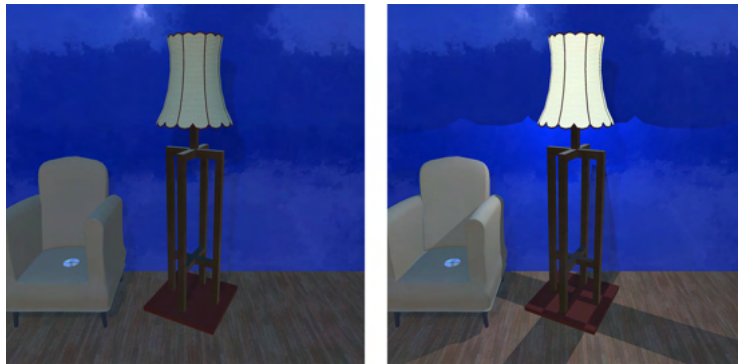
In this work, our goal is to generate diverse and semantically plausible houses. We also aimed to make it easily extendable in the future, adapting to new object types or synthesizing new room types. To this end, we tried to use the best approaches or build on existing works that are insufficient for our use case.



(a) Openness state randomness example with a laptop.



(b) Clean state randomness example with a bed.



(c) On or off state randomness with a floor lamp.

Figure A.22: Examples of object state randomness.

Given the modular nature of our house generation process, if a better algorithm exists at any stage of the pipeline, we can easily update the generation process with that algorithm to generate better houses.

Floorplan Generation. Floorplan generation involves taking a set of rooms to place in a house and an interior boundary (i.e., a top-down outline of the home) and partitioning the interior boundary into rooms. Floorplan generation is a longstanding problem, with many works generating floorplans with procedural generation [146, 11, 209, 70, 153] and deep learning [172, 173, 96, 260]. Our Floorplan generation algorithm is based on [146, 153], which generates a floorplan from a room specification, connectivity constraints between rooms, and an interior boundary. [209] is similar to [146], except that it tries to learn the approximate size of room types from data rather than manually specifying the relative sizes of each room. [96] proposes a similar approach that tries to generate a floorplan based on room preferences, room connectivity constraints, and an interior boundary, but it trains a network on RPLAN [260] to solve these constraints. [172, 173, 260] train a network to generate floorplans, but it does not support inputting any preferences about the number of rooms or the types of rooms in the house. However, in contrast to [146], such work cannot generate arbitrary floorplans that are out of the training distribution, which is problematic if one wanted to generate new types of rooms, such as garages and stairways connecting to another floor [70], or generate massive multi-family floorplans. We also sample an interior boundary for each new house to generate more diverse floorplans.

Object Placement. Object placement involves selecting which objects from a given object database should appear in the house and arranging those objects in a plausible configuration within the rooms of a home (e.g. chairs near tables, paintings on walls, toilets in bathrooms). We built a 3-stage pipeline for placing objects, which (1) places objects on floors, (2) places objects on walls, and (3) places objects on the surface of other objects. Our approach requires specifying remarkably few constraints about how objects are placed in scenes, making it easily extendable to add new objects to our object database and generate new room types.

Many works studying object placement [158, 83, 279, 276, 112, 253, 68, 23, 24, 95, 150] relied on procedural generation. [158, 276, 112, 253] take a given set of objects and the outline of the room but iteratively optimize over several functions to try and minimize the cost function. The cost function determines how realistic the room is with respect to quantities such as how navigable it is and how far an object is from a wall. [279] uses a similar object placement algorithm to ours based on hierarchical relationships between objects. It tries to learn these relationships from 3D-Front [71], whereas we specify constraints and SAGs for objects, such as which can be placed on walls and which objects may appear near each other. In [68], the authors take examples of object arrangements and generate similar ones using probabilistic models trained on 130 scenes. [83] introduces the idea of anchoring objects in parent-child relationships. For instance, objects may be anchored to a wall or on top of a surface such as a table. [23, 24, 95, 150, 20] take in text descriptions or graphs [148] of a furniture arrangement as input and attempt to place objects based on that. However, this work requires manually prompting the model for each new room one wants to generate, so it similarly does not scale well.

Some recent works have proposed using deep learning to place objects on the floors of rooms [245, 248, 183, 205, 280, 30, 71, 139, 179]. The main factors limiting our use of such models are that they: (1) cannot be easily adapted to place novel objects and room types and (2) the lack of high-quality training data of objects placed in 3D scenes. For training data, [245, 248, 205, 280, 139] uses SUNCG [226], a dataset that has been taken down due to legal issues, and [183, 30, 71, 179] uses 3D-Front. However, these approaches do not work with novel objects outside their training dataset and cannot generate novel room types that are not seen during training. Thus, it is impractical to use such approaches in ProctHOR out of the box, as we have differing object databases. It is also impractical (and undesirable) to reproduce such approaches with our object database since we do not have large amounts of training data specifying examples of how our objects are placed in scenes. Here, even if we had such annotations, it would not allow us to add new objects to our object database in the future, as it would require manually collecting

many new examples of where each is placed in scenes to train such models properly. Finally, note that kitchens and bathrooms are not diversely furnished in 3D-Front, so trying to learn object placement in such rooms is impractical.

The approach we use to place objects on surfaces is similar to that of [24, 83], where we calculate the co-object occurrence prior to determine which objects to place on a surface. For example, when looking at which objects to place on a dining table, then the co-object occurrence of a plate is much higher than that of a baseball bat. [113] proposes a fascinating approach to learning co-object occurrences using LLMs [55, 16, 278, 197]. It computes the probability of prompts such as “plate on table” or “baseball bat on table” to compute the relative probabilities of such pairwise combinations. In [68], they propose surface object placement by training probabilistic models that learn to cluster similar objects together as a way to scale much better to new object types. For example, they might input a cluttered desk with a chair, generating many new arrangements of a cluttered desk with new objects on the surface. [249, 69] have done randomizations of objects on surfaces without any priors about which objects should appear on a given surface.

We use semantic asset groups (SAGs) to place co-occurring objects next to each other. We define SAGs in an interactive web environment from top-down images of groups of objects. SAGs are most similar to object arrangements generated from Sketch2Scene [270], which takes an artistic sketch of a scene as input and generates plausible object configurations from that sketch. However, creating the sketches can be incredibly time-consuming, and sampling from them results in a leaky abstraction. In [276, 112, 253], the authors try to place select objects near each other by optimizing a pairwise distance constraint. Here, the cost function is set to minimize the distance between objects, such as chairs and a table, until they are sufficiently close. The relationships between the objects are manually defined at the object type level. SAGs are also similarly related to the idea of hyper relations in [83]. Instead of using positional anchoring, it uses density-based clustering to attempt to sample how objects are anchored around a parent object [206]. In addition, instead of manually defining the hyper-relations, they attempt to extract them from 3D-Front.

A.0.12 Limitations and Future Work

ProcTHOR-10K only generates 1-floor houses. We plan to support multi-floor houses in ProcTHOR-v2.0. This will allow us to capture a wider range of houses and provide better fine-tuning results. Additionally, we plan to scale up our asset databases by leveraging many open-source 3D asset databases, such as ABO [43], PartNet [165], ShapeNet [22], Google Scanned Objects [56], 3D-Future [73], and CO3D [204], among others.

ProcTHOR opens up many avenues of future research in scene synthesis targeted at training embodied agents. Along these lines, better leveraging real-world data as a prior, similar to what is done in Meta-Sim [114], is a promising direction. Similarly, using curriculum learning [176, 171] to train agents in environments that progressively get harder [2] may help train better and faster agents.

A.1 PROCTHOR Datasheet

Motivation	
For what purpose was the dataset created?	The dataset was created to enable the training of simulated embodied agents in substantially more diverse environments.
Who created and funded the dataset?	This work was created and funded by the PRIOR team at Allen Institute for AI. See the contributions section for specific details.
Composition	
What do the instances that comprise the dataset represent?	Each house is specified as a JSON file, which specifies how to populate a 3D Unity scene in AI2-THOR.
How many instances are there in total (of each type, if appropriate)?	There are 10K houses released in the dataset, along with the code to sample substantially more. Section 4 shows the distribution of houses in PROCTHOR-10K.
Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set?	We make 10K houses available, but more houses can easily be sampled with the procedural generation scripts.
What data does each instance consist of?	Each house is specified as a JSON file, which precisely describes how our AI2-THOR build should create the house. The procedurally generated JSON files are typically several thousand lines long.
Is there a label or target associated with each instance?	No.
Is any information missing from individual instances?	No.
Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)?	Each house is generated independently, meaning there are no relationships between the houses.
Are there recommended data splits?	Yes. See Appendix A.0.7 .
Are there any errors, sources of noise, or redundancies in the dataset?	No.
Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)?	The dataset is self-contained.
Does the dataset contain data that might be considered confidential?	No.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety?	No.
---	-----

Collection Process

How was the data associated with each instance acquired?	Each house was procedurally generated. See Appendix ??.
If the dataset is a sample from a larger set, what was the sampling strategy?	The dataset consists of 1 million houses sampled from the procedural generation scripts.
Who was involved in the data collection process?	The authors were the only people involved in constructing the dataset.
Over what timeframe was the data collected?	Data was collected between the end of 2021 and the beginning of 2022.
Were any ethical review processes conducted?	No.

Preprocessing/Cleaning/Labeling

Was any preprocessing/cleaning/labeling of the data done?	Section A.0.7 describes the labeling that was done to make the assets spawn in realistic places. We have also gone through every asset in the asset database to make sure the pivots for each asset are facing a consistent direction.
Was the “raw” data saved in addition to the preprocessed/cleaned/labeled data?	There is no raw data associated with the house JSON files.
Is the software that was used to preprocess/clean/label the data available?	The code to generate the houses is made available.

Uses

Has the dataset been used for any tasks already?	Yes. See the Experiments section of the paper.
--	--

What (other) tasks could the dataset be used for?	<p>The houses can be used in a wide variety of interactive tasks in embodied AI and computer vision.</p> <p>Any task that can be performed in AI2-THOR can be performed in ProcTHOR. For instance, in embodied AI, the houses may be used for navigation [117, 185, 257, 282, 254, 275, 149, 281], multi-agent interaction [102, 103, 2], rearrangement and interaction [249, 74, 79, 37, 228], manipulation [60, 174, 59, 263], Sim2Real transfer [50, 111, 130], embodied vision-and-language [224, 177, 97, 129, 85, 115], audio-visual navigation [34, 78, 33], and virtual reality interaction [258, 169, 93], among others.</p> <p>In the broader field of computer vision, the dataset may be used to study object detection [128]; NeRFs [161, 236, 86, 140]; segmentation, depth, and optimal flow estimation [67, 86]; generative modeling [120, 124, 123]; occlusion reasoning [62]; and pose estimation [29], among others.</p> <p>Our framework for loading in procedurally generated houses from a JSON spec also enables the study of scene clutter generation, building more realistic procedurally generated homes, and the development of synthetically generated spaces to train embodied agents in factories [170], offices, grocery stores [154], and full procedurally generated cities.</p>
Is there anything about the composition of the dataset or the way it was collected and pre-processed/cleaned/labeled that might impact future uses?	No.
Are there tasks for which the dataset should not be used?	Our dataset may be used for both commercial and non-commercial purposes.
Distribution	
Will the dataset be distributed to third parties outside of the entity on behalf of which the dataset was created?	Yes. We plan to make the entirety of the work open-source, including the code used to generate and load houses, the initial static dataset of 1 million procedurally generated house JSON files, and the asset and material databases.
How will the dataset be distributed?	<p>The static house JSON files will be distributed with the PRIOR Python package [51].</p> <p>The code, asset, and material databases will be distributed on GitHub.</p>
Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)?	The house dataset, 3D asset database, and generation code will be released under the Apache 2.0 license.

Have any third parties imposed IP-based or other restrictions on the data associated with the instances?	No.
Do any export controls or other regulatory restrictions apply to the dataset or to individual instances?	No.
Maintenance	
Who will be supporting/hosting/maintaining the dataset?	The authors will be providing support, hosting, and maintaining the dataset.
How can the owner/curator/manager of the dataset be contacted?	For inquiries, email <mattd@allenai.org>.
Is there an erratum?	We will use GitHub issues to track issues with the dataset.
Will the dataset be updated?	We expect to continue adding support for new features to continue to make procedurally generated houses even more diverse and realistic. We also intend to support new tasks in the future.
If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were the individuals in question told that their data would be retained for a fixed period of time and then deleted)?	The dataset does not relate to people.
Will older versions of the dataset continue to be supported/hosted/maintained?	Yes. Revision history will be available for older versions of the dataset.
If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?	Yes. The work will be open-sourced and we intend to provide support to help others use and build upon the dataset.

Table A.1: A datasheet [81] for PROCTOR and PROCTOR-10K.

A.2 ARCHITECTHOR

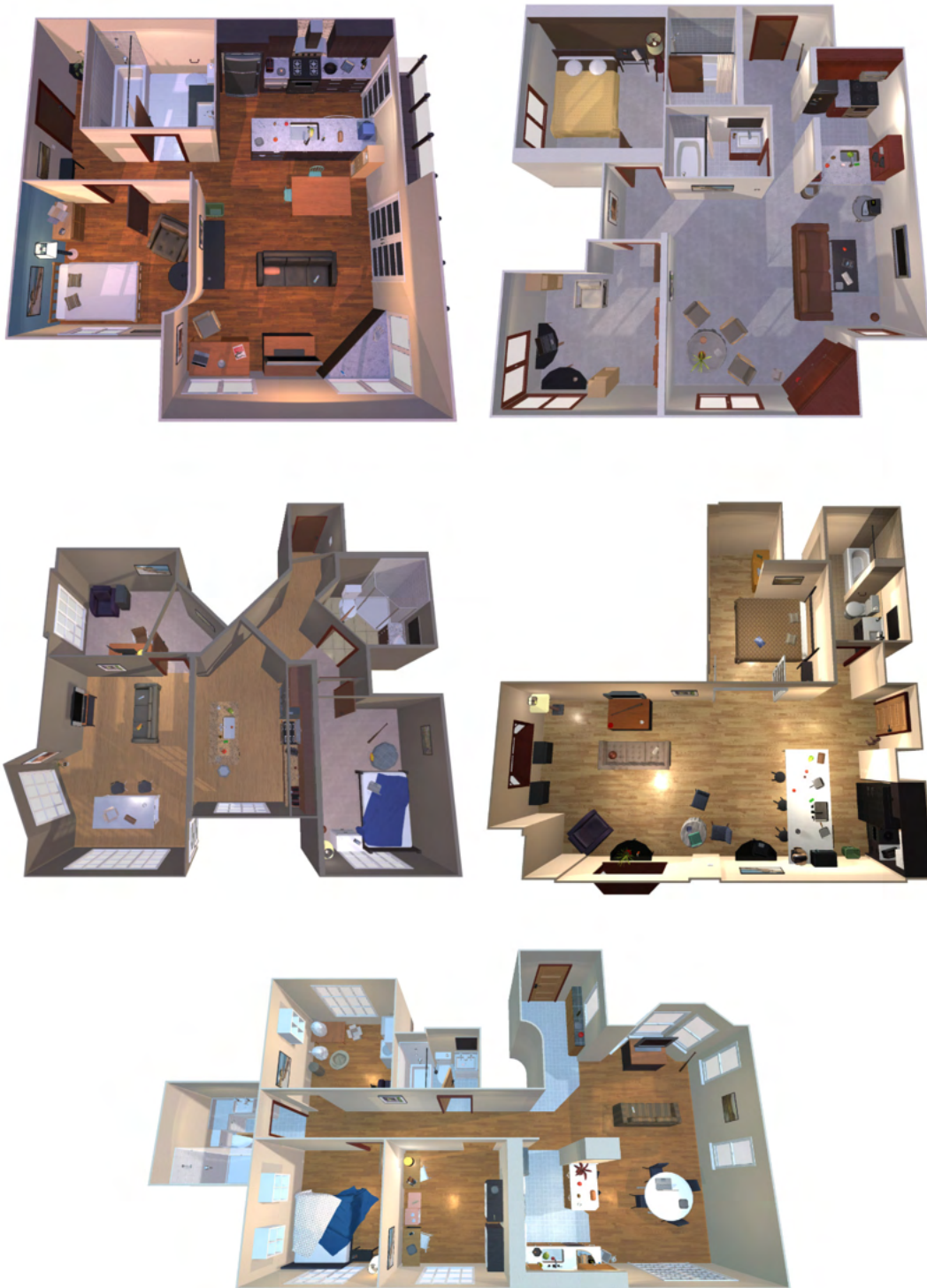


Figure A.23: Top-down images of the 5 custom-built interactive validation houses in ARCHITECTHOR. The goal of these houses is to evaluate interactive agents in more realistic and larger home environments.

A.2.1 Datasheet

Motivation	
For what purpose was the dataset created?	ARCHITECTHOR was created to enable the evaluation of embodied agents in large, realistic, and interactive household environments.
Who created and funded the dataset?	This work was created and funded by the PRIOR team at Allen Institute for AI. See the contributions section for specific details.
Composition	
What do the instances that comprise the dataset represent?	Instances of the dataset comprise interactive 3D houses that were built in Unity and can be used with our custom build of the AI2-THOR API.
How many instances are there in total (of each type, if appropriate)?	There are 10 total houses, comprising 5 validation houses and 5 testing houses.
Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set?	The dataset is self-contained.
What data does each instance consist of?	Each instance of a house is a Unity scene, which includes data such as the placement of objects, lighting, and texturing.
Is there a label or target associated with each instance?	No.
Is any information missing from individual instances?	No.
Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)?	Each house was independently created.
Are there recommended data splits?	Yes. The houses themselves are partitioned as 5 validation houses and 5 testing houses. The assets placed in the house follow the same train/val/test splits used in PROCTHOR-10K.
Are there any errors, sources of noise, or redundancies in the dataset?	No.
Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)?	The dataset is self-contained.
Does the dataset contain data that might be considered confidential?	No.

Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety?	No.
---	-----

Collection Process

How was the data associated with each instance acquired?	Each house was professionally hand-modeled by 3D artists. Most objects placed in the houses come from the PROCTOR asset database. However, countertops, showers, and many cabinets were custom built.
--	---

If the dataset is a sample from a larger set, what was the sampling strategy?	The dataset consists of 1 million houses sampled from the procedural generation scripts.
---	--

Over what timeframe was the data collected?	The houses were built towards the beginning of 2022.
---	--

Were any ethical review processes conducted?	No.
--	-----

Preprocessing/Cleaning/Labeling

Was any preprocessing/cleaning/labeling of the data done?	No.
---	-----

Was the “raw” data saved in addition to the preprocessed/cleaned/labeled data?	There is no raw data associated with the ARCHITECTHOR houses.
--	---

Is the software that was used to preprocess/clean/label the data available?	Yes. We will open-source the ARCHITECTHOR houses and they can be opened and viewed in Unity.
---	--

Uses

Has the dataset been used for any tasks already?	Yes. Please see the Experiments section of the paper.
--	---

What (other) tasks could the dataset be used for?	<p>The tasks can be used for any type of navigation and interaction tasks in embodied AI. The houses are built into our build of AI2-THOR, meaning ARCHITECTHOR can work with any task that can be performed in AI2-THOR.</p> <p>We especially think ARCHITECTHOR will be useful as an evaluation suite for evaluating different sets of PROCTOR tasks and evaluating agents trained on different sets of procedurally generated houses.</p>
---	--

Is there anything about the composition of the dataset or the way it was collected and preprocessed/cleaned/labeled that might impact future uses?	No.
Are there tasks for which the dataset should not be used?	Our dataset may be used for both commercial and non-commercial purposes.
Distribution	
Will the dataset be distributed to third parties outside of the entity on behalf of which the dataset was created?	Yes. All houses in ARCHITECTHOR will be released to the open-source community and available through our build of the AI2-THOR Python API.
How will the dataset be distributed?	The houses will be distributed on GitHub and available to open as Unity scenes.
Will the dataset be distributed under a copyright or other intellectual property (IP) license, and/or under applicable terms of use (ToU)?	ARCHITECTHOR will be released under the Apache 2.0 license.
Have any third parties imposed IP-based or other restrictions on the data associated with the instances?	No.
Do any export controls or other regulatory restrictions apply to the dataset or to individual instances?	No.
Maintenance	
Who will be supporting/hosting/maintaining the dataset?	The authors will be providing support, hosting, and maintaining the dataset.
Is there an erratum?	We will use GitHub issues to track issues with the dataset once it is published.
Will the dataset be updated?	ARCHITECTHOR is currently in maintenance mode and we do not expect it to update much from its current state. However, we plan to actively support future AI2-THOR functionalities in ARCHITECTHOR, such as support for new robots, more advanced interaction capabilities, and bug fixes.
If the dataset relates to people, are there applicable limits on the retention of the data associated with the instances (e.g., were the individuals in question told that their data would be retained for a fixed period of time and then deleted)?	The dataset does not relate to people.

Will older versions of the dataset continue to be supported/hosted/maintained?	Yes. Revision history will be available in the GitHub repository.
If others want to extend/augment/build on/contribute to the dataset, is there a mechanism for them to do so?	Yes. The work will be open-sourced and we intend to provide support to help others use and build upon the dataset.

Table A.2: A datasheet [81] for the artist-designed ARCHITECTHOR houses.

A.2.2 Analysis

ARCHITECTHOR consists of 10 remarkably high-quality large interactive 3D houses. Figure A.23 shows top-down images of the 5 validation houses. Figure A.24 shows some examples of images taken inside of 2 kitchens and a bedroom from ARCHITECTHOR validation.



Figure A.24: Examples of images inside of 2 hand-modeled kitchens and 1 hand-modeled bathroom from ARCHITECTHOR validation.

ARCHITECTHOR was built to be much larger than AI2-iTHOR and RoboTHOR. Figure A.25 shows the size comparisons between comparable hand-built scene datasets in AI2-iTHOR and RoboTHOR, measured in navigable area. Notice that the navigable area in ARCHITECTHOR is substantially larger than in those. The figure also shows the navigable areas in PROCTHOR-10K span the spectrum of navigable areas between AI2-iTHOR, RoboTHOR, and ARCHITECTHOR.

In total, the creation of the 10 houses in ARCHITECTHOR took approximately 320 hours of cumulative work by professional 3D artists. Figure A.26 shows the time breakdown of which parts of the process took the longest. In particular, the creation of custom assets for the kitchen, such as modeling each of the countertops and cabinets, took the longest amount of time, followed by modeling the 3D structure of house.

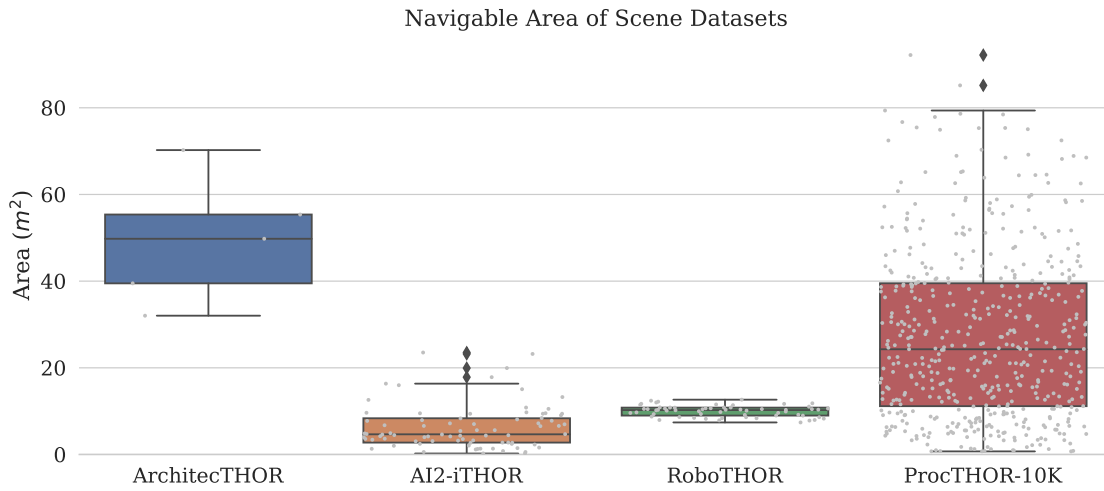


Figure A.25: Box plots of the navigable areas for ARCHITECTHOR compared to AI2-iTHOR, RoboTHOR, and PROCTOR-10K. Validation scenes were used to calculate the data for ARCHITECTHOR, and training scenes were used to calculate the data for AI2-iTHOR, RoboTHOR, and PROCTOR-10K.

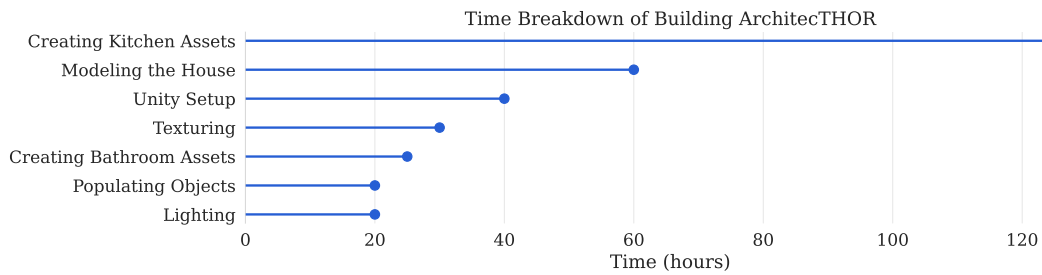


Figure A.26: Cumulative time breakdown of the development of ARCHITECTHOR across 3D artists.

A.3 Input Modalities

A.4 Experiment details

This section discusses the training details used for our experiments. We discuss baselines, PROCTOR pre-training, and environment-specific fine-tuning details for the tasks of ObjectNav, ArmPointNav, and rearrangement.

A.4.1 ObjectNav experiments

For ObjectNav experiments, agents are given a target object type (*e.g.* a bed) and are tasked with finding a path in the environment that navigates to that target object type. The task setup matches what is commonly used in embodied AI [50, 13, 117, 203], although we only utilize forward-facing egocentric RGB images at each time step. All ObjectNav experiments are trained with a simulated LoCoBot (Low Cost Robot) agent [19]. The task and training details are described below.

Evaluation. Following [8], an ObjectNav task is considered successful if all of the following conditions are met:

1. The agent terminates the episode by issuing the `DONE` action.
2. The target object type is within a distance of 1 meter from the agent’s camera.
3. The object is visible in the final frame from the agent’s camera. For instance, if (1) and (2) are satisfied, and the agent is looking in the direction of the object, but the target object is occluded behind a wall, then the task is unsuccessful. Similarly, if the target object type is located in the opposite direction of where the agent is looking, then the task will be unsuccessful.

We also use SPL to evaluate the efficiency of the agent’s trajectory to the target object. SPL is defined and discussed in [8, 13]. A house may have multiple instances of objects for a given type that the agent can successfully reach. For instance, a house may have multiple bedrooms, where each bedroom includes a bed. Here, if the agent navigates to any of the beds, the episode is successful. To calculate SPL in these scenarios, the shortest path length for the task is the minimum shortest path length from the starting position of the agent to any of the reachable target objects of the given type, regardless of which instance the agent navigates towards.

Actions. For each of the trained models, we use a discrete action space consisting of 6 actions, which is shown in Table A.3. Following common practice [50, 111], we use stochastic actuation to better simulate noise in the real world.

Model. We use the relatively simple EmbCLIP [117] training setup for training all ObjectNav experiments. Table A.4 shows the hyperparameters used during training, which are adapted from [117]. Except for the “ProcTHOR+Large” model trained for HM3D (described below), we otherwise use the same model architecture across ObjectNav experiments. Namely, at each time step, the agent receives a $3 \times 224 \times 224$ egocentric RGB image from its camera. The image is processed with a frozen RN50 CLIP-ResNet visual encoder [196] to produce a $2048 \times 7 \times 7$ visual embedding, \mathbf{V}_t . The embedding is compressed through a 2-layer CNN (going from 2048 to 128 to 32 channels) with 1×1 convolutions [232] to obtain a $32 \times 7 \times 7$ tensor, \mathbf{V}'_t .

The target object type is represented as an integer in $\{0, 1, \dots, T\}$, where T is the number of target object types used during training. We use an embedding of t to obtain a 32-dimensional vector. The vector is resized to be a $32 \times 1 \times 1$ tensor. The tensor is then expanded to be of size $32 \times 7 \times 7$, to form our goal target object type embedding \mathbf{G}_t , where the $32 \times 1 \times 1$ tensor is copied 7×7 times.

Action	Description
MOVEAHEAD	Attempts to move the agent forward by $\delta_m \sim \mathcal{N}(\mu = 0.25, \sigma = 0.01)$ meters from its current facing direction. If moving the agent forward by δ_m meters results in a collision in the scene (<i>e.g.</i> there is a wall directly in-front of the agent within δ_m meters), the action fails and the agent’s position remains unchanged.
ROTATERIGHT ROTATELEFT	Rotates the agent rightwards or leftwards from its current forward facing direction by $\delta_r \sim \mathcal{N}(\mu = 30, \sigma = 0.5)$ degrees.
LOOKUP LOOKDOWN	Tilts the agent’s camera up or down by 30 degrees.
DONE	A signal from the agent to terminate the episode and evaluate the trajectory from its current state. Discussed in [8].

Table A.3: The action space for ObjectNav experiments.

We concatenate \mathbf{V}'_t and \mathbf{G}_t to form a $64 \times 7 \times 7$ tensor, which is compressed with a 2-layer CNN to form a $32 \times 7 \times 7$ tensor, \mathbf{Z}_t . The tensor \mathbf{Z} is flattened to form a 1568 dimensional vector, \mathbf{z}_t . Following [174], we use an embedding of the previous action, represented as an integer in $\{0, 1, \dots, 5\}$, to obtain a 6 dimensional vector \mathbf{a}_{t-1} . We concatenate \mathbf{z}_t and \mathbf{a}_{t-1} to form a 1574 dimensional vector \mathbf{x}_t . The vector \mathbf{x}_t is passed through a 1-layer GRU [39, 41] with a hidden belief state \mathbf{b}_{t-1} , of size 512, to obtain \mathbf{b}_t .

Using an actor-critic formulation, the 512-dimensional belief state \mathbf{b}_t is passed through a 1-linear layer, representing the *actor*, to get a 6-dimensional vector, where each entry represents an action. The 6-dimensional vector is passed through a softmax function to obtain the agent’s policy π (*i.e.* the probability distribution over the action space). We sample from π to choose the next action. We also pass the belief state \mathbf{b}_t through a separate 1-linear layer, representing the *critic* to obtain the scalar v , estimating the value of the current state.

The “ProcTHOR+Large” is similar to the above except we: (1) use the larger RN50x16 CLIP-ResNet model, (2) use a 1024-dimensional hidden belief state in our GRU, and (3) input images to the model at a 512×384 resolution.

Hyperparameter	Value
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Value loss coefficient	0.5
Entropy loss coefficient	0.01
Clip parameter (ϵ)	0.1
Rollout timesteps	20
Rollouts per minibatch	1
Learning rate	3e-4
Optimizer	Adam [121]
Gradient clip norm	0.5

Table A.4: Training hyperparameters for ObjectNav experiments.

Training. Each agent is trained using DD-PPO [219, 254], using a clip parameter $\epsilon = 0.1$, an entropy loss coefficient of 0.01, and a value loss coefficient of 0.5. Agents are trained to maximize the cumulative discounted rewards $\sum_{t=0}^H \gamma^t \cdot r_t$, where we set the discount factor γ to 0.99 and the episode’s horizon H to 500 steps. We also employ GAE [218] parameterized by $\lambda = 0.95$.

Reward. The reward function follows that of [117]. Specifically, at each time step, it is calculated as $r_t = \max(0, \min \Delta_{0:t-1} - \Delta_t) + s_t - \rho$, where:

- $\min \Delta_{0:t-1}$ is the minimum L2 distance from the agent to any of the reachable instances of the target object type that the agent has observed over steps $\{0, 1, \dots, t-1\}$.
- Δ_t is the current L2 distance from the agent to the nearest reachable instance of the target object type.
- s_t is the reward for successfully completing the episode. If the agent takes the DONE action and the episode is deemed successful, then s_t is 10. Otherwise, it is 0.
- ρ is the step penalty that encourages the agent to finish the episode quickly. It is set to 0.01.

ProcTHOR pre-training. We pre-train our ObjectNav agents on the full set of 10k training houses in PROCTHOR-10K.¹ We pre-train with all $T = 16$ target object types, which are shown in Table A.5. The agent is trained for 423 million steps, although by 200 million steps, the agent has reached 90% of its peak performance. We used multi-node training to train on 3 AWS g4dn.12xlarge machines, which takes approximately 5 days to complete.

Object Type	RoboTHOR	HM3D-Semantics	AI2-iTHOR	ARCHITECTHOR
Alarm Clock	✓	✗	✓	✓
Apple	✓	✗	✓	✓
Baseball Bat	✓	✗	✓	✓
Basketball	✓	✗	✓	✓
Bed	✗	✓	✓	✓
Bowl	✓	✗	✓	✓
Chair	✗	✓	✓	✓
Garbage Can	✓	✗	✓	✓
House Plant	✓	✓	✓	✓
Laptop	✓	✗	✓	✓
Mug	✓	✗	✓	✓
Sofa	✗	✓	✓	✓
Spray Bottle	✓	✗	✓	✓
Television	✓	✓	✓	✓
Toilet	✗	✓	✓	✓
Vase	✓	✗	✓	✓

Table A.5: The target objects that are used for each ObjectNav task.

¹When training the “ProcTHOR+Large” model used in the HM3D challenge, we use a modified set of 10K houses, see below for details.

Sampling target object types. To sample the target object type for a given episode, we restrict ourselves to only sampling target object types that have a possibility of leading to a successful episode. For instance, even if there is an object like an apple in the scene, it might be located in the fridge, and so if it was used as a target object, the agent would never succeed because the object would never appear visible in the frame (without any manipulation actions). Therefore, we impose a constraint that the target object must be visible without any form of manipulation.

For each house, we use an approximation to determine the set of target object instances that the agent can successfully reach, without any manipulation. Specifically, we start by teleporting the agent into the house, and then perform a BFS over a 0.25×0.25 meter grid to obtain the reachable positions in the scene. A position is considered reachable if teleporting to it would not cause any collisions with any other objects, and the agent is successfully placed on the floor. Then, for each candidate instance of every target object type, we look at the nearest 6 reachable agent positions $\langle x^{(a)}, z^{(a)} \rangle$ to the candidate object instance’s center position. For each reachable agent position, we perform a raycast from the agent’s camera height $y^{(a)}$ to up to 6 random *visibility points* on the object $\langle x^{(o)}, y^{(o)}, z^{(o)} \rangle$. Each object is annotated with visibility points, which are used as a fast approximation to determine if an object is visible with just using a few raycasts, instead of using full segmentation masks. If any of the raycasts from the agent’s reachable position to the object’s visibility point do not have any collisions with other objects (*e.g.* the raycast does not collide with the outside of the fridge), and the L2 distance between $\langle x^{(o)}, y^{(o)}, z^{(o)} \rangle$ and $\langle x^{(a)}, y^{(a)}, z^{(a)} \rangle$ is less than 1 meter, then the object instance is considered successfully reachable by the agent.

To choose a target object type, we use an ϵ -greedy sampling method. Specifically, with a probability of $\epsilon = 0.2$, we randomly sample a target object type that has at least 1 reachable object instance in a given house. With a probability of $1 - \epsilon$, the target object type is the target object type that has been most infrequently sampled in the training process. Since some objects appear much more frequently than others (*e.g.* beds appear in many more houses than baseball bats), sampling based on the least commonly sampled target object types allows us to maintain a more uniform distribution of sampled target object types.

RoboTHOR. RoboTHOR is evaluated in both a 0-shot and fine-tuned setting. For 0-shot, we take the pre-trained model on PROCTHOR-10K and run it on the RoboTHOR evaluation tasks. For fine-tuning, we reduce T to the 12 RoboTHOR target object types, shown in Table A.5 and train on the 60 provided training scenes. We fine-tune for 29 million steps, before validation performance starts to go down, on a machine with 8 NVIDIA Quadro RTX 8000 GPUs. Fine-tuning took about 7 hours to complete.

HM3D-Semantics. We evaluate on HM3D-Semantics in both a 0-shot and fine-tuned setting using the “ProcTHOR” and “ProcTHOR+Large” architectures described above, these two architectures have slightly different pretraining strategies.

“ProcTHOR” model. For 0-shot, we take the pre-trained model on PROCTHOR-10K, and run it on the HM3D-Semantics evaluation tasks. For fine-tuning, we reduce T to the 6 target object types used in HM3D-Semantics (see Table A.5) and train on the 80 provided training houses. We use an early checkpoint from PROCTHOR pre-training, specifically from after 220 million steps. We performed fine-tuning on a machine with 8 NVIDIA RTX A6000 GPUs for approximately 220M steps, which took about 43 hours to complete.

“ProcTHOR+Large” model. We pre-train this model using PROCTHORLARGE-10K a variant of PROCTHOR-10K with houses sampled to better align to the distribution of houses in HM3D. In particular, PROCTHORLARGE-10K contains 10K procedurally generated houses each of which contains between 4 and 10 rooms (houses in PROCTHORLARGE-10K thus tend to be much larger than houses in PROCTHOR-10K). Moreover, during pretraining we only train our agent to navigate to the 6 object

categories used in HM3D-Semantics. Fine-tuning is done identically as above. We use an early checkpoint from PROCTHOR pre-training, specifically from after 125 million steps. We performed fine-tuning on a machine with 8 NVIDIA RTX A6000 GPUs for approximately 185M steps taking 85 hours to complete.

AI2-iTHOR. Similar to RoboTHOR and HM3D-Semantics, we use AI2-iTHOR for both 0-shot and fine-tuning. For 0-shot, we take the pre-trained model on PROCTHOR-10K, and run it on the AI2-iTHOR evaluation tasks. Since the AI2-iTHOR evaluation tasks use the full set of target objects used during PROCTHOR pre-training, we do not need to update T . For fine-tuning, we use a machine with 8 TITAN V GPUs. We fine-tune for approximately 2 million steps before validation performance starts to go down, which takes about 1.5 hours to complete.

ArchitecTHOR. Since ARCHITECThor does not include any training scenes, we only use it for evaluation of the PROCTHOR pre-trained model. As shown in Table A.5, ARCHITECThor evaluation uses the full-set of target object types that are used during PROCTHOR pre-training.

A.4.2 ArmPointNav experiments

In ArmPointNav, we followed the same architecture as [60]. The task is to move a target object from a starting location to a goal location using the relative location of the target in the agent’s coordinate frame. The visual input is encoded using 3 convolutional layers followed by a linear layer to obtain a 512 feature vector. The 3D relative coordinates, specifying the targets, are embedded using three linear layers to a 512 embedding which combined with the visual encoding is input to the GRU. The agent is allowed to take up to 200 steps or the episode will automatically fail.

Hyperparameter	Value
Learning rate	3e-4
Gradient steps	128
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Gradient clip norm	0.5
Rotation Degrees	45
Step penalty	-0.01
Number of RNN Layers	1
Rollouts per minibatch	1
Optimizer	Adam [121]

Table A.6: Training hyperparameters for ArmPointNav experiments.

ProcTHOR pre-training. We pre-train our model on a subset of 7000 houses, on 58 object categories. For each episode, we move the agent to a random location, randomly choose an object in the room that is pickupable, and randomly select a target location. We train our model for 100M frames, running on 4 AWS g4dn.12xlarge machines. Running on a total of 16 GPUs and 192 CPU cores took 3 days of training. Table A.6 shows the hyperparameters used for pre-training.

AI2-iTHOR evaluation. We evaluate our model on 20 test rooms of AI2-THOR (5 kitchens, 5 living rooms, 5 bedrooms, 5 bathrooms), on a subset of 28 object categories for a total of 528 tasks. We attempted to perform fine-tuning on AI2-iTHOR, but none of the fine-tuning models performed better than the zero-shot model trained with PROCTHOR pre-training.

A.4.3 Rearrangement experiments

Following [249, 117], we use imitation learning (IL) to train all models for the 1-phase modality of the task. We divide the full training of the final model into two stages: pre-training in PROCTHOR and fine-tuning in AI2-iTHOR.

Hyperparameter	Value
Rollout timesteps	64
Batch size	7,680
Learning rate	$7.4 \cdot 10^{-4}$
Optimizer	Adam [121]
Gradient clip norm	0.5
BC ^{tf=1} steps	200,000
Dagger steps	2,000,000

Table A.7: ProcTHOR pre-training hyperparameters for Rearrange experiments.

ProcTHOR pre-training. We pre-train our model on a subset of 2,500 one and two-room PROCTHOR-10K houses where a number of 1 to 5 objects are shuffled from their target poses in each episode, including two shuffle modalities: different openness degree (at most one object in an episode) and a different location (up to five objects in an episode). For each house, 20 episodes are sampled such that all shuffled objects are in the same room where the agent is initially spawned. We train with $2 \cdot 10^5$ steps of teacher forcing and 2 million steps of dataset aggregation [208], followed by about 180 million steps of behavior cloning. We use a small set of 200 episodes sampled from 20 validation houses unseen during training to select a checkpoint to evaluate every 5 million steps.

Running on 6 AWS g4dn.12xlarge (totaling 24 GPUs and 288 virtual CPU cores), pre-training with 240 parallel simulations took 4 days. Table A.7 shows the hyperparameters used during pre-training.

AI2-iTHOR fine-tuning. We use the training dataset provided by [6] (4,000 episodes over 80 single-room scenes), and a small subset of 200 episodes from the also provided full validation set to perform model selection. We fine-tune for 3 million steps with 64-step long rollouts, 6 additional million steps with 96-step long rollouts, and another 6 million steps with 128-step long rollouts.

Running on 8 Titan X GPUs and 56 virtual CPU cores, fine-tuning with 40 parallel simulations took 16 hours.

A.5 Performance Benchmark

To calculate the FPS performance benchmark shown in the Analysis section, we partitioned houses into small houses (1-3 room houses) and large houses (7-10 room houses). For the navigation benchmark, we perform move and rotate actions. For the interaction benchmark, we performing a pushing object action. For querying the environment for data, we obtain a piece of metadata from the environment that is not commonly provided at each time step (*e.g.* checking the dimensions of the agent). At each time step, we render a single $3 \times 224 \times 224$ RGB image from the agent’s egocentric perspective. Experiments were conducted on a server with 8 NVIDIA Quadro RTX 8000 GPUs. We employ 15 processes for the single GPU tests and 120 processes for the 8 GPU tests, evenly divided across the GPUs. Table A.8 shows the comparisons to AI2-iTHOR and RoboTHOR.

Compute	Navigation FPS		Isolated Interaction FPS		Environment Query FPS	
	AI2-iTHOR	RoboTHOR	AI2-iTHOR	RoboTHOR	AI2-iTHOR	RoboTHOR
8 GPUs	5,779±189	9,195±294	5,411±190	6,331±137	463,446±18,577	412,550±21,806
1 GPU	1,316±19	1,648±11	1,451±72	1,539±5	169,092±4,232	163,660±3,336
1 Process	180±9	340±26	141±2	217±1	15,584±156	15,578±164
	PROCTOR-S	PROCTOR-L	PROCTOR-S	PROCTOR-L	PROCTOR-S	PROCTOR-L
8 GPUs	8,599±359	3,208±127	6,488±250	2,861±107	480,205±19,684	433,587±18,729
1 GPU	1,427±74	6,280±40	1,265±71	597±37	160,622±2,846	157,567±2,689
1 Process	240±69	115±19	180±42	93±15	14,825±199	14,916±186

Table A.8: Comparing performance benchmarks in PROCTOR to baselines in AI2-iTHOR and RoboTHOR. FPS for navigation, interaction, and querying the environment for data. PROCTOR-S and PROCTOR-L denotes small and large PROCTOR houses, respectively.

A.6 Robustness

We ran ProcTHOR ObjectNav pre-training with 5 different random seeds for 100M steps and found that the variance across seeds is quite small. This measurement was performed for our 0-shot results on a set of 1000 ObjectNav tasks divided evenly between unseen ProcTHOR validation homes, ArchitecTHOR validation scenes, AI2-iTHOR validation scenes, and RoboTHOR validation scenes. Here, we obtained similar performance across all run which had a train success of $67.87\% \pm 2.89\%$ and a val success of $45.3\% \pm 1.2\%$.

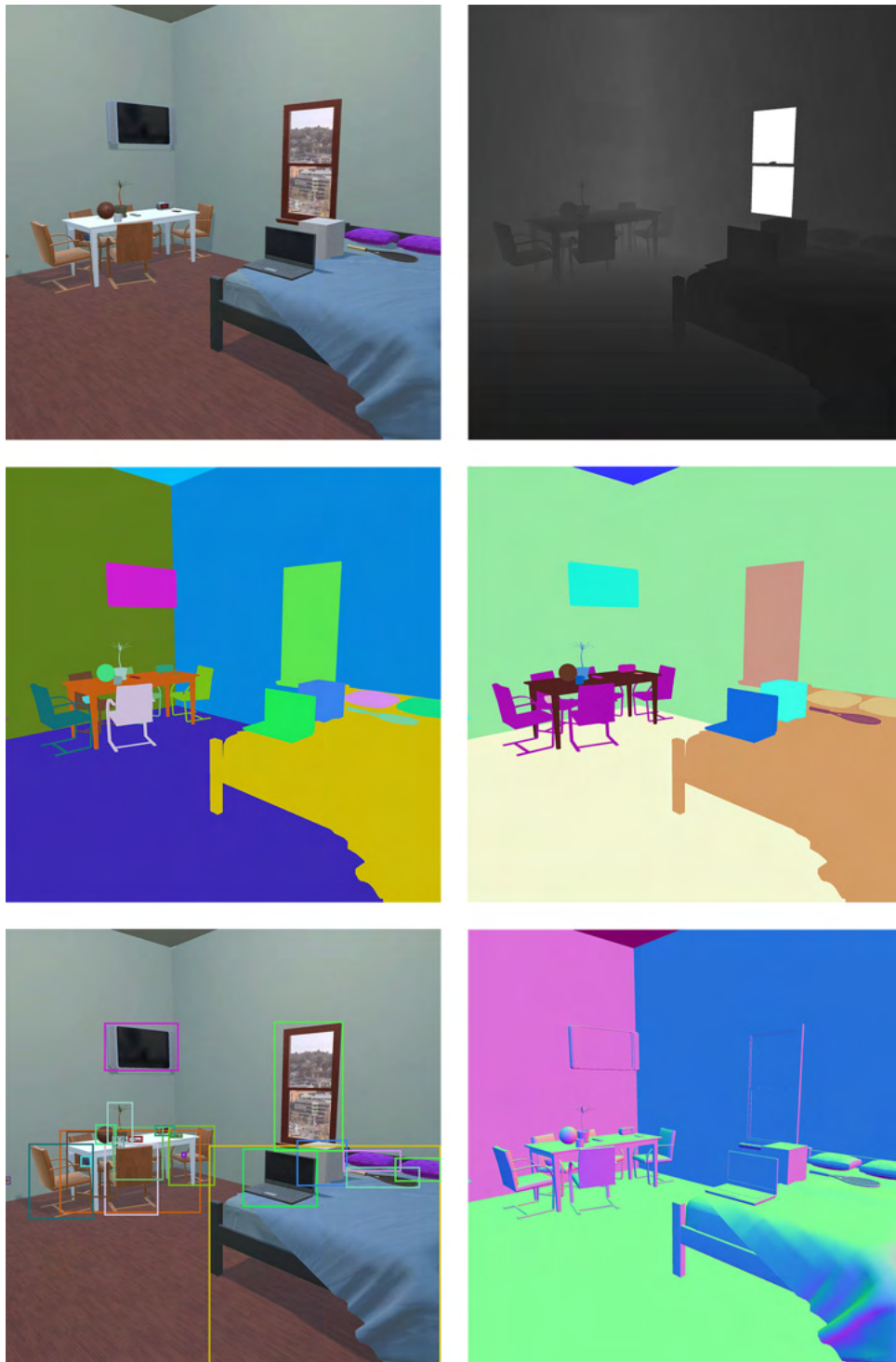


Figure A.27: Examples of image-based modalities available in ProcTHOR include RGB (top left), depth (top right), instance segmentation (middle left), semantic segmentation (middle right), bounding box annotations (bottom left), and surface normals (bottom right). More image modalities can be added by modifying the Unity backend.

Appendix B

Objaverse Appendix

B.1 Instance Segmentation with CP3D

Model. We use the Mask-RCNN[91] model of [5] with a ResNet-50 backbone[92]; no additional changes to their model are made. Instead of a softmax activation, the model uses a Gumbel activation, given by the formula $\eta(q) = \exp(-\exp(-q))$, to transform logits into probabilities. More details about the model and activation can be found in [5].

Training. We take the pretrained ResNet-50 Mask-RCNN checkpoint of [5] and finetune the model for 24 epochs with the CP3D augmentation integrated into the training pipeline. We use a batch size of 64 and a learning rate of 0.002.

Additional Results Here we report detection metrics in addition to the segmentation results reported in the paper in Table B.1. Notably, we see an impressive gain of two points on AP for rare categories.

Method	AP	APr	APc	APf
GOL [5]	27.5	19.8	27.2	31.2
GOL + 3DCP	28.9	21.8	28.7	32.2

Table B.1: **Detection results for bounding box AP category metrics.** APr, APc, and APf measure AP for categories that are rare (appear in 1-10 images), common (appear in 11-100 images), and frequent (appear in >100 images), respectively.

B.2 Open-Vocabulary ObjectNav

Model. The agent’s embodiment is a simulated LoCoBot [19]. The action space consists of six actions: MOVEAHEAD, ROTATELEFT, ROTATERIGHT, END, LOOKUP, and LOOKDOWN. Given the excellent exploration capabilities of EmbCLIP [118, 52], we opt to keep the same overall architecture, just replacing the learned embedding for target types in prior work by a linear projection of the text branch output of CLIP for the target description, as shown in Fig. B.1. Additionally, in order to provide more information about the target and the current visual input, we increase the respective internal representations for each modality from the original 32-D to 256-D. Note that our model does not employ the alternative zero-shot design described in [118], where the target description is not observed by the agent’s RNN. Given the

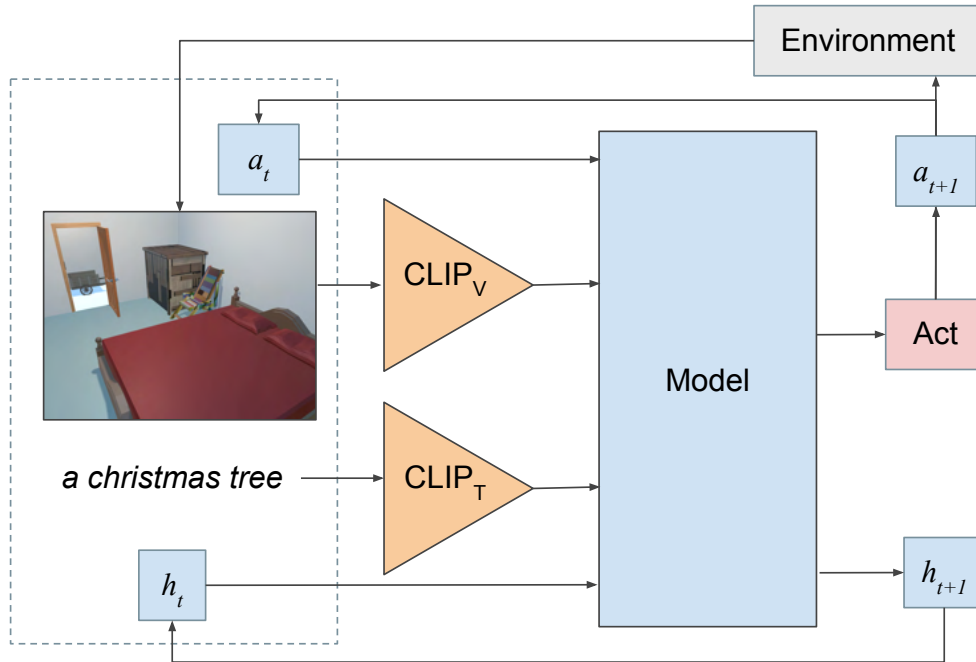


Figure B.1: **Open-Vocabulary ObjNav Model overview.** The ObjectNav model (employing an RNN) uses the high-level architecture illustrated here, where it receives features from the visual and target object description encoders, besides previous hidden units and actions as input, and outputs the next action.

scale of OBJAVERSE-LVIS, we can train agents with good generalization following a more standard design.

Training. For training, we use ProcTHOR to procedurally generate 10,080 houses. Each house has up to three rooms, entirely populated with OBJAVERSE-LVIS assets except for structural components like doors and windows, which are inherited from ProcTHOR [52]. We sample targets corresponding to LVIS categories for which a single instance is present in the scene, resulting in a total of 9,421 unique assets corresponding to 262 categories targeted during training. Training uses DD-PPO [255] and is distributed across 28 GPUs on 7 AWS g4dn.12xlarge machines, with each GPU hosting 360 houses and the subset of OBJAVERSE-LVIS assets populating them. The training hyperparameters are identical to the ones in [52], and the 262 training target categories are listed in Table B.2 and Table B.3, respectively.

Testing. For testing, we sample 150 episodes for each of 30 target categories, which are a subset of the training target categories. The resulting 4,500 episodes are sampled from 151 procedural houses not seen during training. The 30 testing target categories are listed in Table B.4. For the results provided in the main paper, the agent is trained for just 18 million simulation steps, but the resulting policy already shows reasonable performance given the variety of targets and scenes. Improved performance can be achieved with extended training (e.g., after approx. 460 million steps, the success rate is 33.0%).

Hyperparameter	Value
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Value loss coefficient	0.5
Entropy loss coefficient	0.01
Clip parameter (ϵ)	0.1
Rollout horizons	32, 64, 128
Rollout timesteps	20
Rollouts per minibatch	1
Learning rate	$3 \cdot 10^{-4}$
Optimizer	Adam [121]
Gradient clip norm	0.5

Table B.2: Training hyperparameters for Open-Vocabulary ObjectNav.

Bible, Christmas tree, Rollerblade, alligator, ambulance, amplifier, arctic (type of shoe), armor, banner, barbell, barrel, barrow, baseball bat, basketball, bat (animal), bath mat, beachball, bear, bed, beetle, bench, beret, bicycle, binder, binoculars, bird, blackberry, bookcase, boot, bottle, bowling ball, bullhorn, bunk bed, bus (vehicle), butterfly, cab (taxi), cabinet, canoe, cape, car (automobile), card, cardigan, carnation, cart, cassette, cat, chair, chaise longue, chicken (animal), clothes hamper, coatrack, coffee table, cone, convertible (automobile), cornice, cow, cowboy hat, crab (animal), crate, crossbar, cube, cylinder, deck chair, deer, desk, dinghy, dirt bike, dog, dollhouse, doormat, dove, drawer, dresser, duckling, dumbbell, dumpster, easel, elephant, elk, fan, ferret, file cabinet, fireplace, fireplug, fishing rod, flag, flagpole, flamingo, flip-flop (sandal), flipper (footwear), foal, football (American), footstool, forklift, frog, futon, garbage, gargoyle, giant panda, giraffe, golf club, golfcart, gondola (boat), goose, gorilla, gravestone, grill, grizzly, grocery bag, guitar, handcart, hat, heater, hockey stick, hog, horse, horse carriage, jeep, kayak, keg, kennel, kitchen table, kitten, knee pad, ladder, ladybug, lamb (animal), lamp, lamppost, lawn mower, legging (clothing), lion, lizard, locker, log, loveseat, machine gun, mailbox (at home), manhole, mascot, mast, milk can, minivan, monkey, mop, motor, motor scooter, motor vehicle, motorcycle, mushroom, music stool, nut, ostrich, owl, pajamas, parasail (sports), parka, penguin, person, pet, pew (church bench), piano, pickup truck, pinecone, ping-pong ball, playpen, pole, polo shirt, pony, pool table, power shovel, propeller, pug-dog, pumpkin, rabbit, radiator, raincoat, ram (animal), rat, recliner, refrigerator, rhinoceros, rifle, road map, rocking chair, router (computer equipment), runner (carpet), saddle (on an animal), saddle blanket, saddlebag, sandal (type of shoe), scarecrow, scarf, sculpture, seabird, shark, shepherd dog, shield, shirt, shoe, sink, skateboard, ski parka, skullcap, snake, snowmobile, soccer ball, sock, sofa, sofa bed, solar array, sparkler (fireworks), speaker (stereo equipment), spear, spider, sportswear, statue (sculpture), step stool, stepladder, stool, subwoofer, sugarcane (plant), suit (clothing), suitcase, sunhat, surfboard, sweat pants, sweater, swimsuit, table, tape measure, tarp, telephone pole, television camera, tennis ball, tennis racket, tights (clothing), toolbox, tote bag, towel, trailer truck, trampoline, trash can, tricycle, trousers, truck, trunk, turtle, tux, underdrawers, vacuum cleaner, vending machine, vest, wagon wheel, water ski, watering can, wet suit, wheel, window box (for plants), wok, wolf, and wooden leg.

Table B.3: Training target types for Open-Vocabulary ObjectNav.

Christmas tree, bed, bench, blackberry, chair, chicken (animal), dog, easel, elk, fireplug, forklift, garbage, gargoyle, guitar, mascot, motor, penguin, pony, pool table, radiator, rifle, scarf, sock, speaker (stereo equipment), sportswear, sweat pants, trash can, trunk, wet suit, and wheel.

Table B.4: Testing target types for Open-Vocabulary ObjectNav.

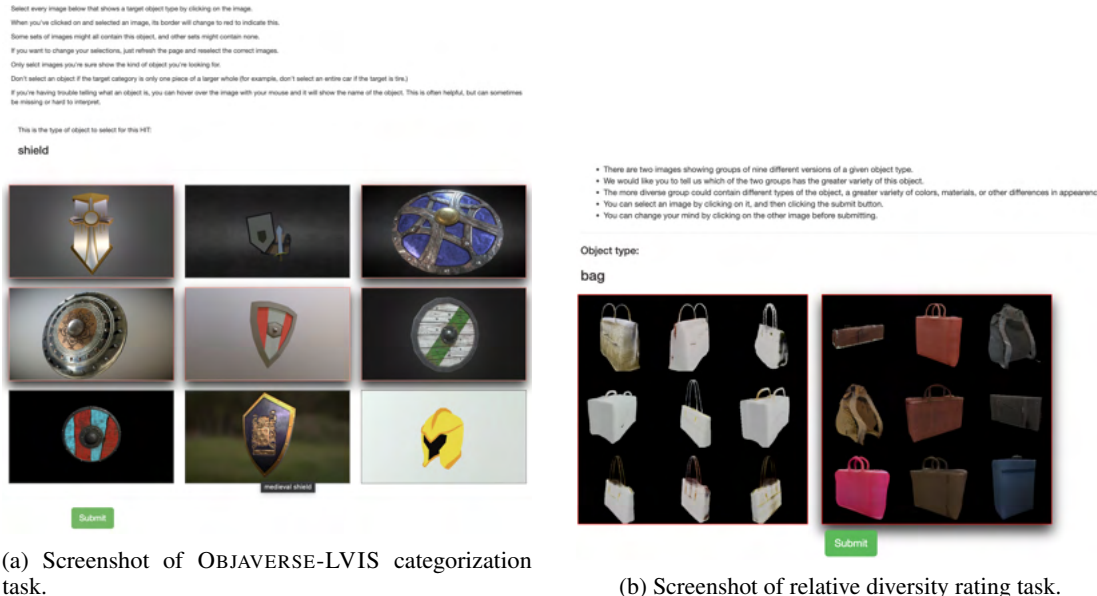


Figure B.2: Data collection interfaces.

B.3 Composition

Human subjects data. A portion of the data included in OBJAVERSE is generated by human subjects (*i.e.* crowdworkers recruited through Amazon’s Mechanical Turk platform) as outlined in Section 3 and detailed below. The collection process has been reviewed and approved for release by an Institutional Review Board.

Data collection interfaces. Human annotators were used to provide the category labels for OBJAVERSE-LVIS as described in Section 3. This task was accomplished by first creating sets of 500 candidate objects for each LVIS category. These candidate sets included objects visually resembling the target category (as ranked by the CLIP features of their thumbnail images), as well as instances whose metadata contained terms with a high similarity to the target category (as ranked by their GloVe vector similarity [184]). Candidate objects were shown to crowdworkers nine at a time, and they were asked to mark objects that were members of the category, as shown in Figure B.2 a. In addition to the visual reference for each object, annotators also had access to the object’s name and were encouraged to use this when helpful. Human annotators were also used to rate the relative diversity of 3D objects generated by models trained using OBJAVERSE and ShapeNet. The user interface and instructions for this task are shown in Figure B.2 b. Two sets of nine objects generated by each model were shown with random left-right orientations, and workers were asked to choose the set exhibiting the greater variety in appearance.

B.4 Estimating Coverage

We use OpenAI’s CLIP ViT-B/32 model to estimate the categorical coverage of the objects in OBJAVERSE. Specifically, for each object, we compute the CLIP image embedding from the thumbnail and the cosine similarity between an text embedding of each WordNet entity [65]. The entity with the maximum cosine similarity is then assigned as the object’s entity. The WordNet entities are textually encoded in the form, “a {entity} is a {definition}”, which is loosely inspired by CuPL [191]. For instance, we might have “a *bat* is a nocturnal mouselike mammal with forelimbs modified to form membranous wings and anatomical adaptations for echolocation by which they navigate” or “a *bat* is a club used for hitting a ball in various games”. Computing the nearest WordNet entity for each object gave us an estimated coverage of 20.8K entities.

Appendix C

Phone2Proc Appendix

C.1 Implementation Details

For all experiments, we use the same architectures and process from EmbCLIP [119] and adopt the same hyperparameters as ProcTHOR [52]. The 3x224x224 RGB images are processed with a frozen CLIP-ResNet-50 architecture [92, 195]. This visual embedding is compressed with a 2-layer CNN, concatenated with a goal object type embedding, and compressed with a 2-layer CNN. This is flattened and combined with an embedding of the previous action, then passed through a single-layer GRU [41] policy with a hidden belief state of size 512. An actor and critic are used to generate the next action probability distribution and current state value estimates, respectively. The agent’s next action is sampled from the actor distribution.

The following updates from [52] are made for policy and goal encoder fine-tuning:

1. Learning rate is lowered to $3e-5$ (10x lower than that of ProcTHOR [52]).
2. A small penalty of -0.05 is assessed if the agent runs into objects.
3. If the agent is about to run into an object, it will randomly move and rotate in small increments to coarsely emulate unmodeled and unintended physical environment interactions.
4. The neck actions are limited to looking 30° above and below the horizon, as on our physical platforms.
5. The horizontal field of view for a fixed aspect ratio is randomized by episode (uniformly sampled in 0.2° increments between 48° and 65°), and the vertical field of view/aspect ratio is modified to more closely resemble the Intel RealSense D435.

We use multi-node training on 3 or 4 (depending on the environment size) AWS g4dn.12xlarge machines with 16 processes per machine.

The Habitat baseline used in Table 1 is trained on the 80 HM3D [198] set training scenes used for the 2022 Habitat challenge [272] using 80 processes and 8 A100 GPUs. The model trained for 200M steps and we used the model which achieved the best performance on a validation set of 200 episodes. It was trained with the same updated field of view as every other model and baseline evaluated in this work.

C.2 Failure Cases

The most common failure case for PHONE2PROC models was semantic confusion; that is, not being able to recognize particular instances of objects or mistaking instances of other object categories for the target object. For example, a couch with a cover on it in the 6-room apartment was mistaken for a bed several times in the limited field of view of the agent and spare jugs for the water cooler in the cafeteria were mistaken as a vase. To generate the scenes, only six 3D models of vases were used. Thus, some semantic confusion is perhaps unsurprising, and PHONE2PROC with more visual diversity might be used to even greater effect.