A. Scene Graph Details.

A.1. Details about relationships in scene graphs.

As mentioned in Section 3.1, we use the edges of the scene graph to encode various kinds of relationships. Note that all these relationships are directed, therefore are not commutative in nature. We define three **Agent-Object** relationships, namely *Sees*, *Holds* and *Touches*. *Sees* encodes whether an object is currently within 2.5 meters and in the field of view of the egocentric camera on the agent. *Holds* indicates whether an object is picked up by the agent. We only use it with the arm based agent [7] for ArmPointNav. Similarly, *Touches* is also applicable when using an arm-based agents. It includes objects that the agent has picked up, and also the ones that its arm might have collided with.

We have three **Object-Object** relationships, namely *On*, *Near* and *Adjacent*. An object is considered *Near* another object, if the distance between them is below a certain threshold. However, for an object to be *Adjacent* to another, they need to be *Near* and also have an unobstructed path between their centers. We also have **Agent-Conditioned Object-Object** spatial relationships that include *Right*, *Left*, *Above*, *Below*, *Front*, and *Behind*. These relationships are determined based on the each object's position relative to the agent's coordinate frame. Lastly we define a relationship *Contains* which can be an Object-Object, Room-Object or a Room-Agent relationship. For instance, if the agent is present in the kitchen, the relationship "Kitchen *Contains* Agent" would be true.

To summarise, we define the following relationships between various nodes of the graph:

- Sees
- Near
- Above
- Contains
- Touches
- Adjacent
- Below
- Holds
- Right
- Front
- On
- Left
- Behind

B. Experiment Details.

In this section, we provide the training and hyperparameter settings for all our models on Object Navigation (ObjNav), Multi-Object Navigation (MultiON) and ArmPointNav.

B.1. ObjectNav.

For ObjectNav, an agent is tasked to find a target object type (e.g. bed) in an environment. We provide the target object type in the form of an embedding, and use forward-facing egocentric RGB images at each time step. All our ObjectNav models are trained with a simulated LoCoBot [1] agent. The action space consists of MoveAhead, RotateLeft, RotateRight, LookUp, LookDown, and End. The rotation degree for Rotate and Look actions is 30°. We describe the task and training details below.

Evaluation. Following [2], an ObjectNav task is considered successful if:

- The agent terminates the episode by calling the End action.
- An instance of the target object type is within a distance of 1 meter from the agent.
- The object is visible from the agent's camera. If the object is occluded behind an obstacle or out of the agent's view, the episode is considered unsuccessful.

We also use SPL [2, 3] to evaluate the efficiency of our agents. If an environment has multiple instances of the target object type, if the agent navigates to any of those instances, it is considered successful. For calculating the SPL in such scenarios, the shortest path is defined as the minimum shortest path length from the starting position to any of the reachable instance of the given type, regardless of which instance the agent navigates towards.

We evaluate *zero-shot* on RoboTHOR [4], AI2-iTHOR [10] and ArchitecTHOR [6] benchmarks. To reiterate, *zero-shot* here implies that no training was performed on any of these benchmarks. Table 1 shows the object types used for respective benchmarks. We use the PRIOR package [5] for loading datasets for all benchmarks.

Training. Each agent is trained using DD-PPO [15], using a clip parameter $\epsilon = 0.1$, an entropy loss coefficient of 0.01, and a value loss coefficient of 0.5. We use the reward structure and model architecture from [8, 6]. We summarize the training hyperparameters in Table 2.

As mentioned in the main paper, we train our ObjectNav agents in *ProcTHOR-train* environments from the ProcTHOR-10k dataset [6]. We train with 16 target object types shown in Table 1. We train with 48 parallel processes on 8 NVIDIA RTX A6000 GPUs for 350 million steps. All the ObjectNav models we present took 6-8 days to train depending on whether they had an auxiliary objective or not. We follow the strategy proposed in [6] to sample target object types during training. Additionally, as we remark in Section 4.1 (line 478), we present training curves for Object Navigation training in Figure 1.

B.2. Multi-ObjectNav.

In Multi-ObjectNav (MultiON), an agent needs to find multiple target objects in a particular order. We perform experiments with two variants of this task, MultiON-2 and MultiON-3 that requires an agent to find 2 and 3 target objects in an episode, respectively. The action space contains MoveAhead, RotateLeft, RotateRight, LookUp, LookDown and Found actions.

Evaluation. We collect a validation set in 200 *ProcTHOR-Val* houses using the same set of target objects presented in Table 1. The agent needs to issue a Found action to indicate if it found the requested object. We build upon the success criteria for ObjectNav to define a successful Found

Object Type	RoboTHOR	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 1: **Target object types used for each ObjectNav benchmark.**

Hyperparameter	Value
Discount factor (γ)	0.99
GAE [13] 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 [9]
Gradient clip norm	0.5

Table 2: **Training hyperparameters for ObjectNav.**

Hyperparameter	Value
Discount factor (γ)	0.99
GAE [13] parameter (λ)	0.95
Number of RNN Layers	1
Step penalty	-0.01
Gradient Steps	128
Rollouts per minibatch	1
Learning rate	3e-4
Optimizer	Adam [9]
Gradient clip norm	0.5

Table 3: **Training hyperparameters for ArmPointNav.**

action as:

• An instance of the current target object type is within a distance of 1 meter from the agent.

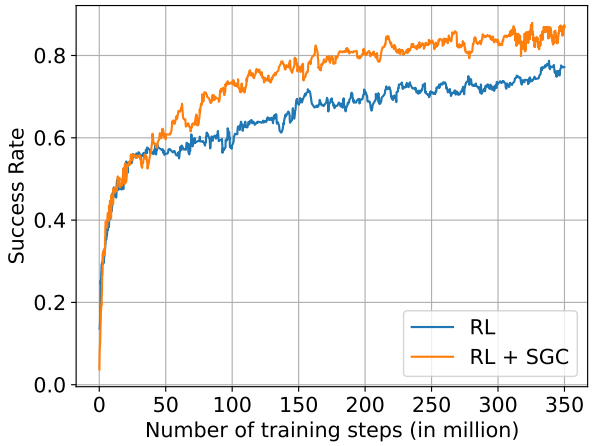


Figure 1: **ObjectNav training Plot.** We present the training curves for ObjectNav training for RL and RL + SGC models.

• The current target object is visible from the agent's camera. If the target object is occluded or out of the agent's view, the FOUND action is considered unsuccessful.

For a MultiON-2 episode to be successful, the agent requires to execute 2 successful FOUND actions. A failed FOUND action at any point in the episode renders it as a failure. We extend the same criteria to 3 objects for MultiON-3.

Training. Similar to ObjectNav, we train in *ProcTHOR-train* environments from the ProcTHOR-10k dataset [6]. We use the 16 target object types presented in Table 1, and sample 2 or 3 object types for each episode based on the

Multi-ON variant. We extend the object sampling strategy from [6] to multiple objects to ensure roughly uniform sampling of target object types. At the beginning of each episode, we provide the first target object, and provide the next target object only after a successful FOUND action. This allows us to use the EmbodiedCLIP [8] Object Navigation architecture. We use the same distance based reward shaping from [6, 8], and provide a reward of 10.0 for each successful FOUND action.

Each MultiON agent is trained with DD-PPO [15] using the same set of hyperparameters as ObjectNav presented in Table 2. We train with 48 parallel processes on 8 NVIDIA RTX A6000 GPUs for 180 million steps. Training takes 2 days with just RL and 5 days with the RL + SGC baseline.

B.3. ArmPointNav.

For ArmPointNav, we follow the same architecture as [7] with the exception of using a CLIP-pretrained frozen ResNet-50 encoder as our visual backbone instead of a 3-layer CNN. Our motivation for this design choice was improved results for the RL model with the CLIP backbone. Therefore, we use CLIP-pretrained ResNet-50 for both RL and RL + SGC models. We use the manipulathor [7] agent, which is equipped with a kinova arm. The action space for the ArmPointNav agent includes three navigation actions (MOVEAHEAD, ROTATELEFT, and ROTATERIGHT), with rotations of 45 degrees. It also has 8 arm-based actions, namely MOVE-ARM-{X,Y,Z}-{P,M}, which allow the wrist to move in a plus (P) or minus (M) direction relative to the agent along the (X,Y,Z) axis, and MOVE-ARM-HEIGHT-{P,M}, which modifies the arm height.

Evaluation. We evaluate our model on the validation set from [7]. The task requires the agent 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. It only uses egocentric RGB observation as its visual input.

Training. Following [6], we train our ArmPointNav models on a subset of 7,000 houses with 58 object categories. For each training episode, we teleport the agent to a random location, randomly sample a pickupable object, and randomly sample a target location. We train our models for 90 million steps on 8 NVIDIA RTX A6000s. We list the training hyperparameters in Table 3. Training takes 80 hours with just RL and 100 hours with RL + SGC. We note that SGC is applied to 20% of the steps in a batch.

B.4. Adapting to Novel Object Categories.

As presented in Section 4.5.2 in the main draft, we show that models trained with SGC are able to adapt to novel object categories much faster. We provide the 10 sets of 5 Object Categories sampled for this experiment in Table 4. We also provide the larger pool of 80+ categories that they were sampled from in Table 5.

Set1	Pot, Egg, CellPhone, CD, ToiletPaper
Set2	CellPhone, Faucet, Fork, Desktop, Box
Set3	Newspaper, Lettuce, ButterKnife, Spatula, CellPhone
Set4	Lettuce, Faucet, CoffeeMachine, Pot, Stool
Set5	TeddyBear, SideTable, Lettuce, DeskLamp, CoffeeMachine
Set6	TVStand, Safe, Desktop, Ottoman, Tomato
Set7	WashingMachine, Toaster, SideTable, Plunger, Desk
Set8	Drawer, Statue, Toaster, Fridge, Newspaper
Set9	Microwave, FloorLamp, Sink, DiningTable, Stool
Set10	Drawer, SinkBasin, LaundryHamper, DeskLamp, Sink

Table 4: **Sets of randomly sampled object categories.**

C. Linear Probing Details.

Data capture. We collect data from about 5 episodes in each of approx. 2,500 ProcTHOR-10K [6] houses, using an exploration policy that iteratively visits the nearest unvisited reachable object in the house until no more objects are left to visit. During the trajectory, we store the current scene graph, both RL and RL + SGC agents' beliefs, and environment metadata like agent pose and scene name.

Processing pipeline. For each probing experiment, given the collected beliefs and task-dependent binary labels, we subsample the collected data to produce a compact set of training samples. In order to improve numerical stability, we compress the beliefs via PCA, ensuring 99% of the data variance is explained. We fit a logistic regression model on the compressed training dataset, and evaluate on an (also compressed) held-out test set. The used implementations for PCA and logistic regression are from [12].

Reachability. For these experiments, we define reachability as whether a location at an Euclidean distance R from the agent and with relative orientation θ is "free-space" (*i.e.*, can be occupied by the agent without collision). The free space of a scene is estimated by applying marching squares [11, 14] with a threshold level of 0.9999 to its binary grid of reachability (with $\{0, 1\}$ indicating {occupied, free} space). We consider three concentric circumferences around the agent's current location and locate points on these circumferences that are at 30° spaced angles with respect to the agent's current orientation, resulting in 36 unique relative locations. Each of the 36 relative locations is used as a target variable. In these experiments we use 20,000 data points for each of the 36 target variables, of which 18,000 are for training and 2,000 for testing. We sample these data points so that negative and positive samples are balanced for each target variable, both for training and for testing. All accuracy metrics provided for this task are thus "balanced accuracy" rather than standard accuracy. We fit the 36 models using the processing pipeline described above.

Visibility. In these experiments we subsample 40,000 datapoints of which 33,000 are used for training, 2,000 for validation, and 5,000 for testing. In order to estimate an agent's

ArmChair, Book, Boots, Bottle, Box, Bread, ButterKnife, CD, Cabinet, Candle, Cart, Cellphone, Cloth, ClothesDryer, CoffeeMachine, CoffeeTable, CounterTop, CreditCard, Cup, Desk, DeskLamp, Desktop, DiningTable, DishSponge, DogBed, Doorframe, Doorway, Drawer, Dresser, Dumbbell, Egg, Faucet, Floor, FloorLamp, Fork, Fridge, GarbageBag, Kettle, KeyChain, Knife, Ladle, LaundryHamper, Lettuce, Microwave, Newspaper, Ottoman, Painting, Pan, PaperTowelRoll, Pen, Pencil, PepperShaker, Pillow, Plate, Plunger, Pot, Potato, RemoteControl, RoomDecor, Safe, SaltShaker, Shelf, ShelvingUnit, SideTable, Sink, SinkBasin, SoapBar, SoapBottle, Spatula, Spoon, Statue, Stool, TVStand, TableTopDecor, TeddyBear, TennisRacket, TissueBox, Toaster, ToiletPaper, Tomato, VacuumCleaner, Wall, WashingMachine, Watch, Window, WineBottle

Table 5: **List of All Object Types.**

understanding of visibility, we record, at every sampled location, which objects where visible to the agent within a 2.5 meters range. As most objects will not be visible at any given agent position, naïvely sampling the 40,000 datapoints above would result in many objects having almost no positive (*i.e.* visible) examples. Because of this, we use an iterative sampling procedure that prioritizes selecting samples which contain instances of visible objects that are otherwise underrepresented in the dataset. Even using this iterative approach there is significant class imbalance with objects of type RoomDecor being visible in only 2% of datapoints (the minimum across all object types) while the Painting objects are visible in 20% (the maximum across all object types). For this reason, when we fit separate logistic regression models to predict visibility for each of these object types we reweigh samples so as to ensure that the negative and positive examples have, in total, equal importance. When computing accuracy metrics we thus also use the "balanced accuracy" rather than the standard accuracy which would overweigh negative examples in this setting. In the main paper we also report results when fitting models exactly as above but using, as target, whether or not an object was visible at the current timestep or any previous time step in the episode so far. These additional results give insight into the agents' ability to remember having seen objects in the past.

Revisited state detection. In addition to the two probing experiments described in the main paper, we also probe for the detection of revisited states. We quantize the current agent pose relative to the one at the beginning of the episode using square cells of $0.25 \times 0.25\ m^2$ for location and arcs of 30 degrees for rotation. We treat each pair of relative grid location and rotation as a separate state. In these experiments we use 20,000 data points, of which 18,000 are for training and 2,000 for testing. The RL + SGC agent achieves 56.20% balanced accuracy, whereas the RL-only agent reaches 52.10% on this task.

D. Loss Behavior.

To show the SGC loss predictions, we present a plot of prediction probabilities for our ObjectNav RL + SGC model

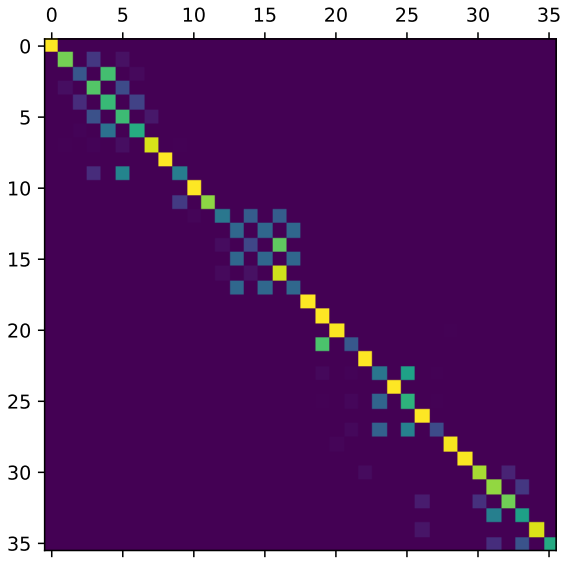


Figure 2: **Graph Loss Plot.**

in Figure 2. We show the plot with 18 rollout steps across 2 parallel processes *i.e.* with a total batch size of 36. Note how the predictions are concentrated around the diagonal which is akin to the ground truth present in Figure 3 of the main paper. It is also interesting to see some deviations from the diagonal, which shows that there is some scope to optimize the loss even better and potentially lead to more performance benefits. These deviations arise from scene graphs from nearby time steps in the same episode being very slightly different, making it harder to classify them.

E. ObjectNav Qualitative Analysis.

We visualize some trajectories and provide qualitative analysis on how SGC is affecting the ObjectNav behavior. We attach the videos and trajectory maps with the supplementary zip file. Each trajectory map shows the top down

view of the path traversed by the agent. It also indicates the target object location by a purple box. We indicate the target object type and the number of steps on the bottom left corner of the frame in the trajectory videos.

ARCHITECTHOR. Firstly we show some top-down trajectories in ARCHITECTHOR environments (Fig. 3), noting the floor plans' large scale and photorealism. Example 1 shows the ObjectNav agents trying to find a spray bottle. The RL-only model looks into one part of the house, then terminates the trajectory unsuccessfully, whereas RL + SGC is able to navigate to the bathroom and successfully finds the target.

Example 2 shows an example where the RL-only agent falsely recognizes the microwave as a television. The RL + SGC agent also starts by checking the kitchen and the microwave, then turns around and explores another region and is able to find the television successfully.

Example 3 shows an interesting failure case for both RL and RL + SGC models. The target object is a vase. The RL-only agent looks for the vase in the area around the dining table and then terminates the trajectory failing to find the target. The trajectory map shows the limited exploration of this model.

On the other hand, even though the RL + SGC agent fails to find the target, it explores the room in a much more exhaustive fashion as can be seen in the respective trajectory map. It exhausts the maximum number of steps allowed and times out instead of falsely calling an END action.

RoboTHOR. We show three qualitative examples from the RoboTHOR benchmark as well (Fig. 4). They all seem to highlight the inability of the RL-only agent to explore the scene, which leads to unsuccessful termination of the episode. On the other hand, the RL + SGC agent explores the region it starts in and then goes to different parts of the scene, which eventually allows it to succeed.

References

[1] Carnegie mellon university. locobot: an open source low cost robot. *http://www.locobot.org/*. 1

[2] 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. 1

[3] 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. 1

[4] 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. 1

[5] Matt Deitke, Aniruddha Kembhavi, and Luca Weihs. PRIOR: A Python Package for Seamless Data Distribution in AI Workflows, 2022. 1

[6] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. 2022. 1, 2, 3

[7] 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. 1, 3

[8] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings for embodied ai. *CVPR*, 2022. 1, 3

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 2

[10] 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. 1

[11] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In Maureen C. Stone, editor, *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987*, pages 163–169. ACM, 1987. 3

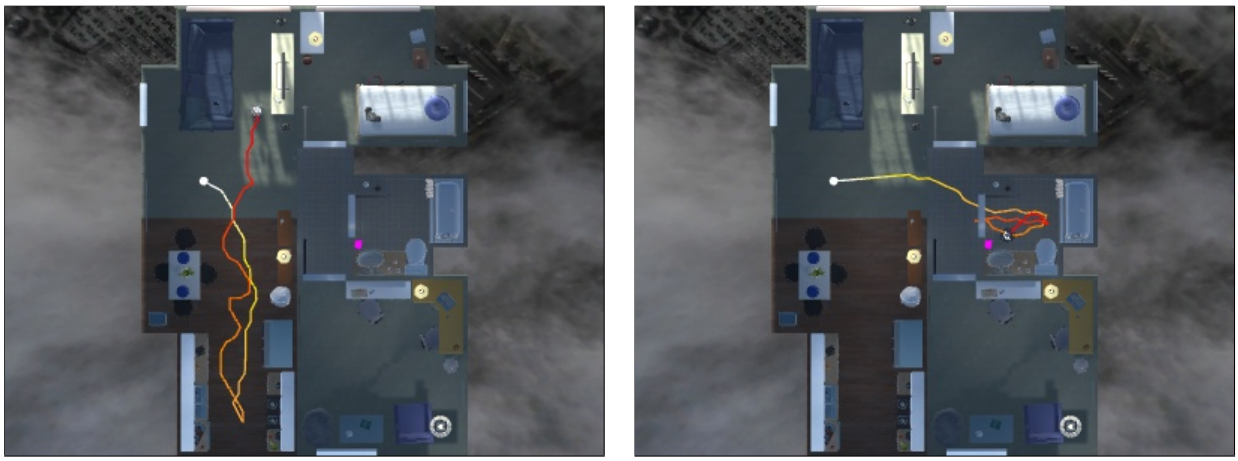
[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 3

[13] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *ICLR*, 2016. 2

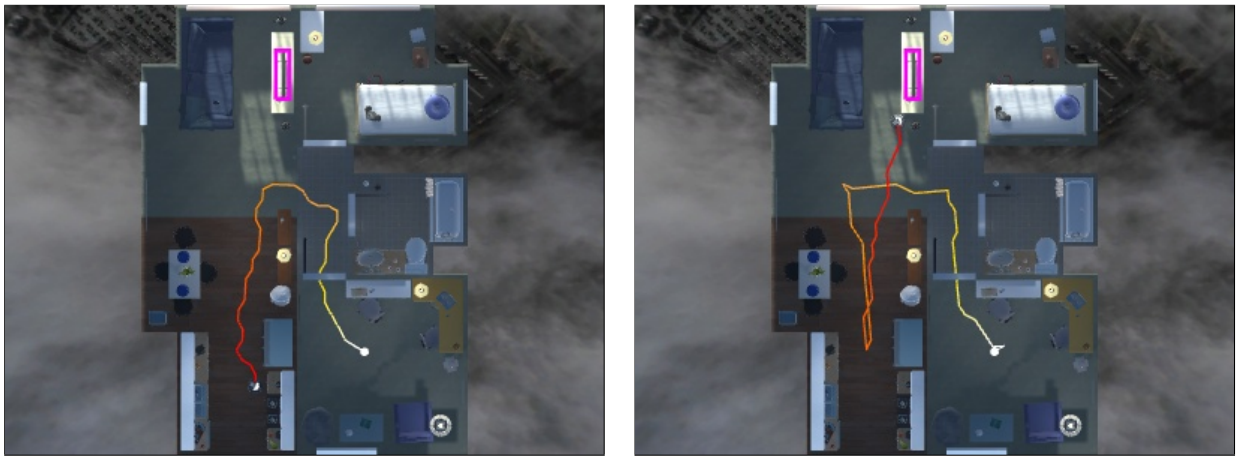
[14] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: Image processing in python. *CoRR*, abs/1407.6245, 2014. 3

[15] 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. ICLR, 2020. 1, 3

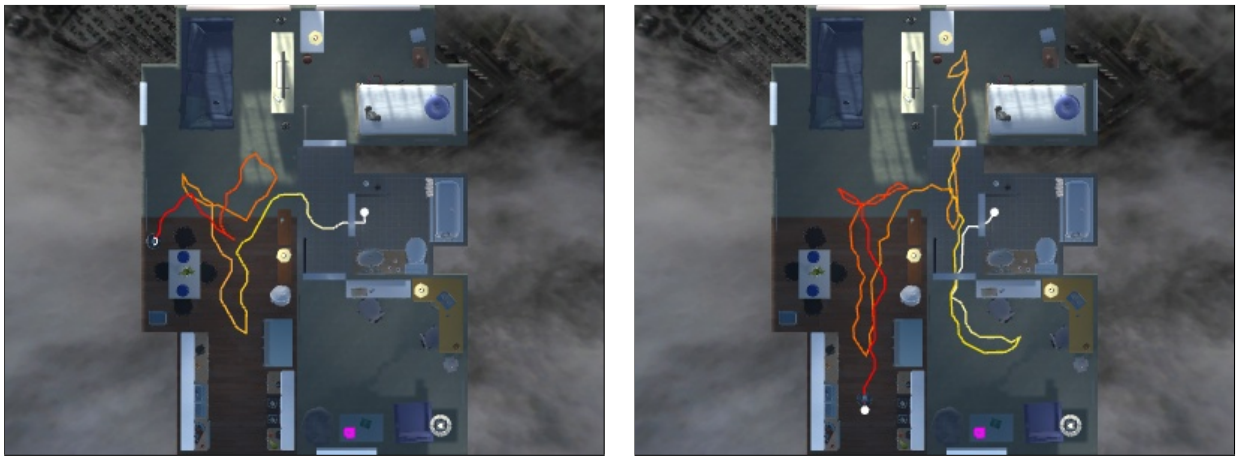
RL-only RL + SGC



Example 1. Target object type: Spray bottle.



Example 2. Target object type: Television.



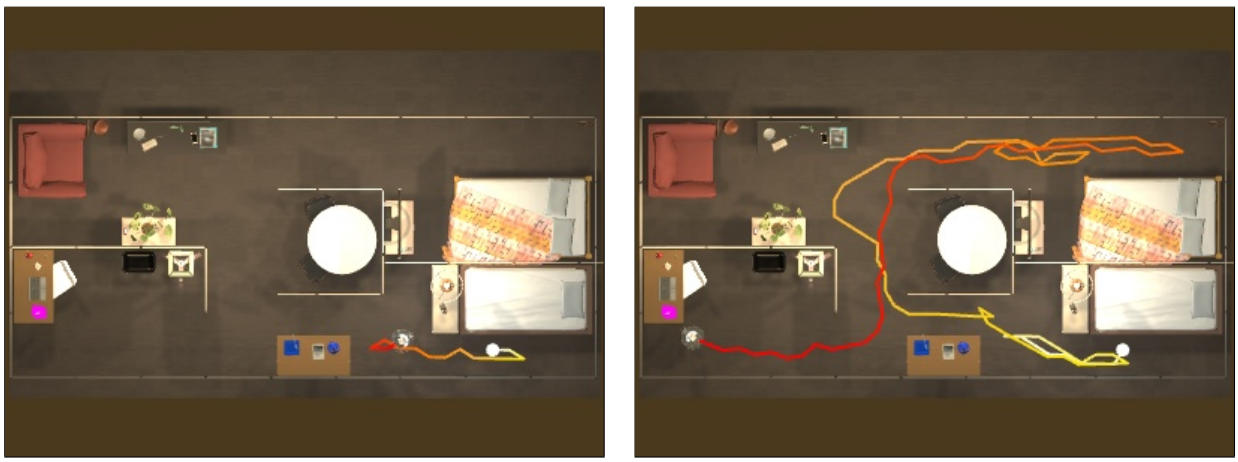
Example 3. Target object type: Vase.

Figure 3: **Top-down trajectories for RL-only and RL + SGC agents in ARCHITECTHOR.** Examples 1 and 2 correspond to successful trajectories for the RL + SGC agent, whereas Example 3 shows different behaviors for failed episodes. The target object type in each episode is highlighted in magenta. For the trajectory lines, {white, red} correspond to the {beginning, end} of each episode.

RL-only RL + SGC

Example 1. Target object type: Spray bottle.

Example 2. Target object type: Baseball bat.



Example 3. Target object type: Mug.

Figure 4: **Top-down trajectories for RL-only and RL + SGC agents in RoboTHOR.** The target object type in each episode is highlighted in magenta. For the trajectory lines, {white, red} correspond to the {beginning, end} of each episode.