

Visual Room Rearrangement

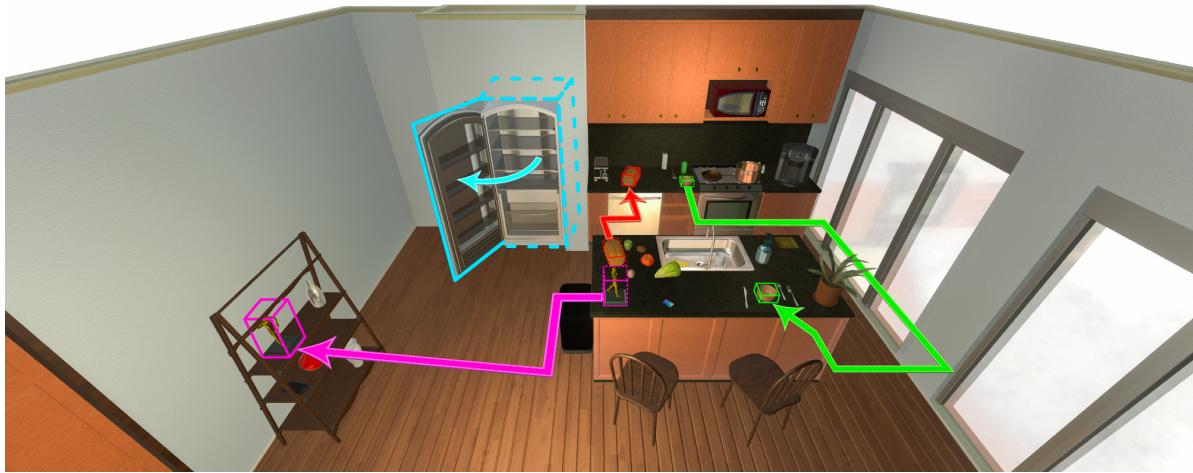
Luca Weihs¹Matt Deitke^{1,2}Aniruddha Kembhavi^{1,2}Roozbeh Mottaghi^{1,2}¹PRIOR @ Allen Institute for AI ²University of Washingtonai2thor.allenai.org/rearrangement

Figure 1: An instance of the Room Rearrangement task. Objects begin in the positions indicated by the solid 3D bounding boxes. An agent must walk through the room and record the objects it sees. The agent is then removed, and objects are moved to the locations indicated by the dashed bounding boxes. The agent is then reintroduced into the room and must interact with objects (moving or opening them) to return the room to its original state.

Abstract

There has been a significant recent progress in the field of Embodied AI with researchers developing models and algorithms enabling embodied agents to navigate and interact within completely unseen environments. In this paper, we propose a new dataset and baseline models for the task of Rearrangement. We particularly focus on the task of Room Rearrangement: an agent begins by exploring a room and recording objects’ initial configurations. We then remove the agent and change the poses and states (e.g., open/closed) of some objects in the room. The agent must restore the initial configurations of all objects in the room. Our dataset, named RoomR, includes 6,000 distinct rearrangement settings involving 72 different object types in 120 scenes. Our experiments show that solving this challenging interactive task that involves navigation and object interaction is beyond the capabilities of the current state-of-the-art techniques for embodied tasks and we are still very far from achieving perfect performance on these types of tasks.

1. Introduction

One of the longstanding goals of Embodied AI is to build agents that interact with their surrounding world and perform tasks. Recently, navigation and instruction following tasks have gained popularity [1, 2, 4] in the Embodied AI community. These tasks are the building blocks of interactive embodied agents, and over the past few years, we have observed remarkable progress regarding the development of models and algorithms. However, a typical assumption for these tasks is that the environment is static; namely, the agent can move within the environment but cannot interact with objects or modify their state. The ability to interact with and change its environment is crucial for any artificial embodied agent and cannot be studied in static environments. There is a general trend towards interactive tasks [50, 41, 49]. These tasks focus on specific aspects of interaction such as object manipulation, long-horizon planning and understanding pre-condition and post-conditions of actions. In this paper, we address a more comprehensive task in a visually rich environment that can subsume each of these skills.

We address an instantiation of the *rearrangement* problem, an interactive task, recently introduced by Batra et al. [3]. The goal of the rearrangement task is to reach a goal room configuration from an initial room configuration through interaction. In our instantiation, an agent must recover a scene configuration after we have randomly moved, or changed the state of, several objects (*e.g.* see Fig. 1).

This problem has two stages: *walkthrough* and *unshuffle*. During the walkthrough stage, the agent may explore the scene and, through egocentric perception, record information regarding the goal configuration. We then remove the agent from the room and move some objects to other locations or change their state (*e.g.* opening a closed microwave). In the unshuffle stage, the agent must interact with objects in the room to recover the goal configuration observed in the walkthrough stage.

Rearrangement poses several challenges such as inferring the visual differences between the initial and goal configurations, inferring the objects’ state, learning the post-conditions and pre-conditions of actions, maintaining a persistent and compact memory representation during the walkthrough stage, and successful navigation. To establish baseline performance for our task, we evaluate an actor-critic model akin to the state-of-the-art models used for long-horizon tasks such as navigation. We train our baselines using decentralized distributed proximal policy optimization (DD-PPO) [47, 40], a reward-based RL algorithm, as well as with DAgger [37], a behavioral cloning method. During the walkthrough stage, the agent uses a non-parametric mapping module to memorize its observations along with any visible objects and their positions. In the unshuffle stage the agent compares images that it observes against what it has observed in its map and may use this information to inform which objects it should move or open. As a proof-of-concept we also run experiments with a model that includes a semantic mapping component adapted from the Active Neural SLAM model [8].

To facilitate research in this challenging direction, we compiled the Room Rearrangement (RoomR) dataset. RoomR is built upon AI2-THOR [29], a virtual interactive environment that enables interacting with objects and changing their state. The RoomR dataset includes 6,000 rearrangement tasks that involve changing the pose and state of multiple objects within an episode. The level of the difficulty of each episode varies depending on the differences between the initial and the goal object configurations. We have used 120 rooms and more than 70 unique object categories to create the dataset.

We consider two variations of the room rearrangement task. In the first setting, which we call the *1-Phase* task, the agent completes the walkthrough and unshuffle stages in parallel so that it is given aligned images from the walkthrough and unshuffle configurations at every step. In the

second setting, the *2-Phase* task, the agent must complete the walkthrough and unshuffle stages sequentially; this *2-Phase* variant is more challenging as it requires the agent to reason over longer time spans. Highlighting the difficulty of the rearrangement, our evaluations show that our strong baselines struggle even in the easier *1-Phase* task. Rearrangement poses a new set of challenges for the embodied-AI community. Our code and dataset are publicly available. A supplementary video¹ provides the description of the task and some qualitative results.

2. Related Work

Embodied AI tasks. In recent years, we have witnessed a surge of interest in learning-based Embodied AI tasks. Various tasks have been proposed in this domain: navigation towards objects [4, 51, 48, 7] or towards a specific point [1, 38, 47, 36], scene exploration [9, 8], embodied question answering [18, 13], task completion [55], instruction following [2, 41], object manipulation [16, 52], multi-agent coordination [24, 23], and many others. Rearrangement can be considered as a broader task that encompasses skills learned through these tasks.

Rearrangement. *Rearrangement Planning* is an established field in robotics research where the goal is to reach a goal state from an initial state [5, 44, 27, 31, 53, 33]. While these methods have shown impressive performance, they consider complete observability of the state from perfect visual perception [11, 27], a planar surface as the environment [30, 42], a static robot [15, 32], same environment for evaluation of generalization [39, 26], or a limited set of object categories or limited variation within the categories [10, 19]. Some works address some of these issues, such as generalization to new objects or imperfect perception [54, 6]. In this paper, we take a step further and relax these assumptions by considering raw visual input instead of perfect perception, a visually and geometrically complex scene as the configuration space, separate scenes for training and evaluation, a variety of objects, and object state changes.

Task and motion planning. Our work can be considered as an instance of joint task and motion planning [25, 43, 35, 17, 12] since solving the rearrangement task requires low-level motion planning to plan a sequence of actions and high-level task planning to recover the goal state from the initial state of the scene. However, the focus of these works is primarily on the planning problem rather than perception.

3. The Room Rearrangement Task

3.1. Definition

Our goal is to rearrange an initial configuration of a room into a goal configuration. So that our agent does not have to

¹<https://youtu.be/1APxaOC9U-A>

reason about soft-body physics, we restrict our attention to piece-wise rigid objects. Suppose a room contains n piece-wise rigid objects. We define the state for object i as $s_i = (p_i, o_i, c_i, b_i)$ where

- $p_i \in \{3D \text{ rotations and translations}\} = \text{SE}(3)$ records the pose of the object,
- if the object can be opened $o_i \in [0, 1]$ specifies the *openness* of an object (*e.g.* $o_i = 0.5$ means a door is half open) and if the object cannot be opened (*e.g.* a mug) then $o_i = \emptyset$,
- c_i records the 8 coordinates in \mathbb{R}^3 of the corners of the 3D bounding box for object i , and
- $b_i \in \{0, 1\}$ records if the i th object is “broken” (1 if broken, otherwise 0).

While this definition of an object’s state is constrained (*e.g.* objects can be more than just “broken” and “unbroken”) it matches well the capabilities of our target embodied environment (AI2-THOR) and can be easily enriched as embodied environments become increasingly realistic. We now let $S = \text{SE}(3) \times ([0, 1] \cup \{\emptyset\}) \times \mathbb{R}^{8 \cdot 3} \times \{0, 1\}$ be the set of all possible poses for a single object and $\mathcal{S} = \prod_{i=1}^n S$ the set of all possible joint object poses. The agent’s goal is to convert an initial configuration $s^0 \in \mathcal{S}$ to a goal $s^* \in \mathcal{S}$.

Our task has two stages: (1) *walkthrough* and (2) *unshuffle*. During the walkthrough stage, the agent is placed into a room with goal state s^* , and it should collect as much information as needed for that particular state of the room in a maximum number of actions (for us, 250). The agent is removed from the room after the walkthrough stage. We then select a random subset of the n objects and change their state. The state change may be a change in p or o . This state will be the initial state s^0 that the agent observes at the beginning of the unshuffle stage. The agent’s goal is to convert s^0 to s^* ($s^0 \rightarrow s^*$) via a sequence of actions.

3.2. Metrics

To quantify an agent’s performance, we introduce four metrics below. Recall from the above that an agent begins an unshuffle episode with the room in state s^0 and has the goal of rearranging the room to end in state s^* . Suppose that at the end of an unshuffle episode, the agent has reconfigured the room so that it lies in state $s = (s_1, \dots, s_n) \in \mathcal{S}$. In practice, we cannot expect that the agent will place objects in exactly the same positions as in s^* . We instead choose a collection of thresholds which determine if two object poses are, approximately, equal. When two poses (s_i, s_i^*) are approximately equal we write $s_i \approx s_i^*$. Otherwise we write $s_i \not\approx s_i^*$.

Let $s_i^1, s_i^2 \in S$ be two possible poses for object i . As it makes little intuitive sense to compare the poses of broken objects, we will always assert that poses of broken objects are unequal. Thus if $b_i^1 = 1$ or $b_i^2 = 1$ we define $s_i^1 \not\approx s_i^2$. Now let’s assume that neither $b_i^1 = 1$ nor $b_i^2 = 1$. If ob-

ject i is pickupable, let $\text{IOU}(s_i^1, s_i^2)$ be the intersection over union between the 3D bounding boxes c_i^1, c_i^2 . We then say that $s_i^1 \approx s_i^2$ if, and only if, $\text{IOU}(s_i^1, s_i^2) \geq 0.5$. If object i is openable but not pickupable, we say that $s_i^1 \approx s_i^2$ if, and only if, $|o_i^1 - o_i^2| \leq 0.2$. The use of the IOU above means that object poses can be approximately equal even when their orientations are completely different. While this can be easily made more stringent, our rearrangement task is already quite challenging. Note also that our below metrics do not consider the case where there are multiple identical objects in a scene (as this does not occur in our dataset). We now describe our metrics.

Success (SUCCESS) – This is the most unforgiving of our metrics and equals 1 if all object poses in s and s^* are approximately equal, otherwise it equals 0.

% Fixed (Strict) (%FIXEDSTRICT) – The above SUCCESS metric does not give any credit to an agent if it manages to rearrange some, but not all, objects within a room. To this end, let $M_{\text{start}} = \{i \mid s_i^0 \not\approx s_i^*\}$ be the set of misplaced objects at the start of the unshuffle stage and let $M_{\text{end}} = \{i \mid s_i \not\approx s_i^*\}$ be the set of misplaced objects at the end of the episode. We then let %FIXEDSTRICT equal 0 if $|M_{\text{end}} \setminus M_{\text{start}}| > 0$ (*i.e.* the agent has moved an object that should not have been moved) and, otherwise, let %FIXEDSTRICT equal $1 - |M_{\text{end}}| / |M_{\text{start}}|$ (*i.e.* the proportion of objects that were misplaced initially but ended in the correct pose).

% Energy Remaining (%E) – Missing from all of the above metrics is the ability to give partial credit if, for example, the agent moves an object across a room and towards the goal pose, but fails to place it so that it has a sufficiently high IOU with the goal. To allow for partial credit, we define an energy function $D : S \times S \rightarrow [0, 1]$ that monotonically decreases to 0 as two poses get closer together (see the Appendix E for full details) and which equals zero if two poses are approximately equal. The %E metric is then defined as the amount of energy remaining at the end of the unshuffle episode divided by the total energy at the start of the unshuffle episode, *e.g.* $\%E = (\sum_{i=1}^n D(s_i, s_i^*)) / (\sum_{i=1}^n D(s_i^0, s_i^*))$.

Changed (#CHANGED) – To give additional insight as to our agent’s behavior we also include the #CHANGED metric. This metric is simply the the number of objects whose pose has been changed by the agent during the unshuffle stage. Note that larger or smaller values of this metric are not necessarily “better” (both moving no objects and moving many objects randomly are poor strategies).

The above metrics are then averaged across episodes when reporting results.

4. The RoomR Dataset

The Room Rearrangement (RoomR) dataset utilizes 120 rooms in AI2-THOR [29] and contains 6,000 unique rearrangements (50 rearrangements per training, validation, and

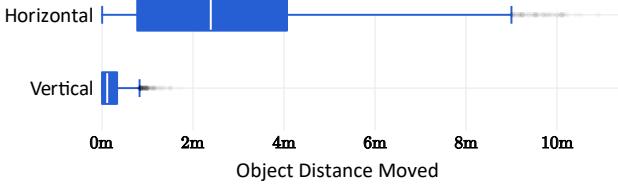


Figure 2: **Distance distribution.** The horizontal (Manhattan distance) and vertical distance distributions between changed objects in their goal and initial positions.

testing room). Each datapoint consists of an initial room state s_0 , the agent’s starting position, and the goal state s^* .

4.1. Generating Rearrangements

The automatic generation of the dataset enables us to scale up the number of rearrangements easily. We generate each room rearrangement using the procedure that follows.

Place agent. We randomize the agent’s position on the floor. The position is restricted to lie on a grid, where each cell is of size $0.25\text{m} \times 0.25\text{m}$. The agent’s rotation is then randomly chosen amongst $\{0^\circ, 90^\circ, 180^\circ, 270^\circ\}$. The agent’s starting pose is the same for both s^0 and s^* .

Shuffle background objects. To obtain different configurations of objects for each task in the dataset, we randomly shuffle each movable object, ensuring background objects do not always appear in the same position. Shuffled objects are never hidden inside other receptacles (*e.g.* fridges, cabinets), which reduces the task’s complexity.

Sample objects. We now randomly sample a set of $N \geq 0$ openable but non-pickupable objects and a set of $M \geq 0$ pickupable objects. These objects and counts are chosen randomly with $N \in \{0, 1\}$ and $M \in \{1 - N, \dots, 5 - N\}$.

Goal (s^*) setup. We open the N objects sampled in the last step to some randomly chosen degree of openness in $[0, 1]$ and move the other M pickupable objects to arbitrary locations within the room. The room’s current state is now s^* , the start state for the walkthrough stage.

Initial (s^0) setup. We randomize the N sampled openable objects’ openness and shuffle the position of each of the M sampled pickupable objects once more. We are now in s^0 , the start state for the unshuffle stage.

In the above process, we ensure that no broken objects are in s^0 or s^* . While we provide a fixed number of datapoints per room, this process can be used to sample a practically unbounded number of rearrangements.

4.2. Dataset Properties

Rooms. There are 120 rooms across the categories of kitchen, living room, bathroom, and bedroom (30 rooms for each category). We designate 20 rooms for training, 5 rooms for validation, and 5 rooms for testing, across each room category. Of the 6,000 unique rearrangements in our dataset, 4000 are designated for training, 1000 are set in

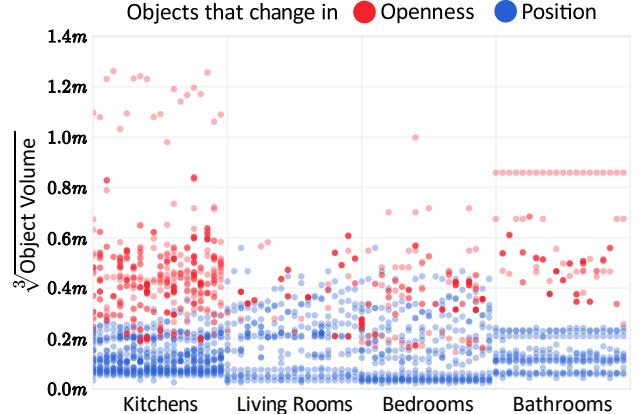


Figure 3: **Distribution of object size.** Each column contains the cube root of every object’s bounding box volume that may change in openness (red) or position (blue) for a particular room. Notice that, across room categories, objects that change in position are significantly smaller than objects that change in openness.

validation rooms, and 1000 are set in test rooms. For each such split, there are 50 rearrangements per room.

Objects. There are 118 object categories (listed in Appendix F), among which 62 are pickupable (*e.g.* cup) and 10 are openable and non-pickupable (*e.g.* fridge). The set of object categories that appear in the validation and testing rooms is a subset of the object categories that appear during training. Thus, if a plant appears in a validation or testing room, then a plant is also present in one of the training rooms. While all object categories are seen during training, the physical appearance of object instances are often unique in training, validation, and testing rooms. AI2-THOR provides annotation as to if an object is pickupable, openable, movable, or static.

Across the dataset, there are 1895 pickupable object instances and 1262 openable non-pickupable object instances (an average of 15.7 and 10.5, respectively, per room). Fig. 2 shows the distance distribution (horizontal and vertical) of objects between their initial and goal positions. It illustrates the complexity of the problem, where the agent must travel relatively far to recover the goal configuration. Fig. 3 shows the distribution of these object groups and their sizes within every room. Note that pickupable objects (*e.g.* apple, fork) tend to be relatively small and hard to find, compared to openable non-pickupable objects (*e.g.* cabinets, drawers). Further, across room categories, the number of openable non-pickupable objects varies considerably.

5. Model

In our experiments, Sec. 6, we consider two RoomR task variants: *1-Phase* and *2-Phase*. In the *1-Phase* task, the agent completes the unshuffle and walkthrough stages si-

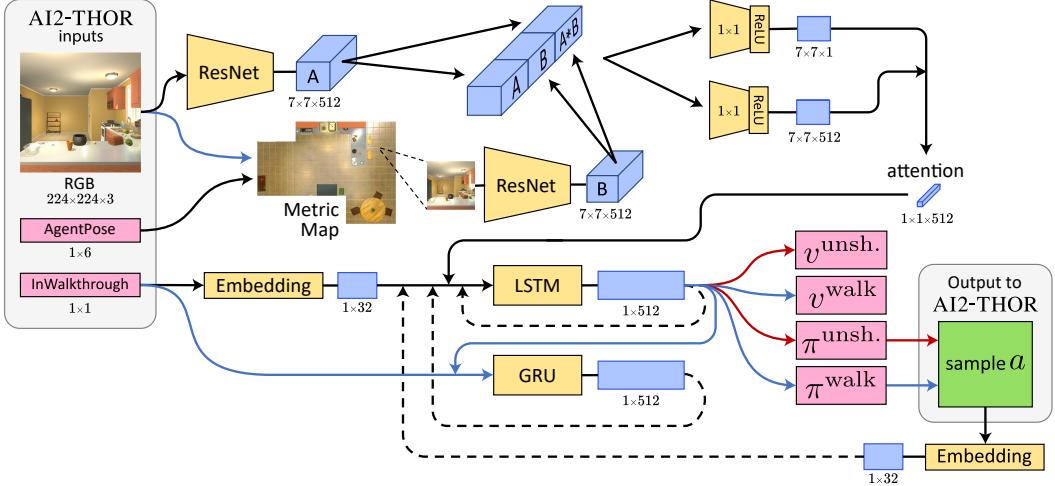


Figure 4: **Model overview.** The model is used for both the unshuffle and walkthrough stages. The connections specific to the walkthrough and unshuffle stages are shown in blue and red, respectively. The dashed lines represent connections from the previous time step. The model’s trainable parameters, inputs and outputs, and intermediate features are shown in yellow, pink, and blue, respectively.

multaneously in lock step. The model we employ for this *1-Phase* task is a simplification of the model used when performing the *2-Phase* task (in which both stages must be completed sequentially and so longer-term memory is required). For space we only describe the *2-Phase* model below, see our codebase for all architectural details.

Our network architecture, see Fig. 4, follows the same basic structure as is commonly employed within Embodied AI tasks [38, 47, 14, 45, 23]: a combination of a convolutional neural network to process input egocentric images, a collection of embedding layers to encode discrete inputs, and an RNN to enable the agent to reason through time. In addition to this baseline architecture, we would like our agent to have two capabilities relevant to the rearrangement task, namely the abilities to, during the unshuffle stage, (a) explicitly compare images seen during the walkthrough stage against those seen during the unshuffle stage, and (b) reference an implicit representation of the walkthrough stage. We now describe the details of our architecture and how we enable these additional capabilities.

Our agents are of the actor-critic [34] variety and thus, at each timestep $t \geq 0$, given observations ω_t (e.g. an egocentric RGB image) and a summary h_{t-1} of the agent’s history, we require that an agent produces a policy $\pi_\theta(\omega_t | h_{t-1})$ (i.e. a distribution over the agent’s actions) and a value $v_\theta(\omega_t | h_{t-1})$ (i.e. an estimate of future rewards). Here we let $\theta \in \Theta$ be a catch-all parameter representing all of the trainable parameters in our network. As we wish for our agent to have characteristically different behavior in the walkthrough and unshuffle stages, we have two separate policies π_θ^{walk} and $\pi_\theta^{\text{unsh.}}$ (and similarly for v_θ).

To encode input $224 \times 224 \times 3$ RGB egocentric images, we use a ResNet18 [21] model (pretrained on ImageNet)

with frozen model weights with the final average pooling and classification layers removed. This ResNet18 model transforms input images into $7 \times 7 \times 512$ tensors. For our RNN, we leverage a 1-layer LSTM [22] with 512 hidden units. To produce the policies π^{walk} and $\pi^{\text{unsh.}}$ we use two 512×84 linear layers, each applied to the output from the LSTM, and each followed by a softmax nonlinearity. Similarly, to produce the two values v^{walk} and $v^{\text{unsh.}}$ we use two distinct 512×1 linear layers applied to the output of the LSTM with no additional nonlinearity. We now describe how we enable agents the abilities (a) and (b) above.

Mapping and image comparison. Our model includes a non-parametric mapping module. The module saves the RGB images seen by the agent during the walkthrough stage, along with the agent’s pose. During the unshuffle stage, the agent (i) queries the metric map for all poses visited during the walkthrough stage, (ii) chooses the pose closest to the agent’s current pose, and then (iii) retrieves the image saved by the walkthrough agent at that pose. Using an attention mechanism, the agent can then compare this retrieved image against its current observation to decide which objects to target.

Implicit representations of the walkthrough stage. In addition to explicitly storing the images seen during the walkthrough stage, we also wish to enable our agent to produce an implicit representation of its experiences during the walkthrough stage. To this end, at every timestep t during the walkthrough stage we pass h_t , the output of the 1-layer LSTM described above, to a 1-layer GRU with 512 hidden units to produce the walkthrough encoding w_t . During the unshuffle stage this walkthrough encoding is no longer updated and is simply taken as the encoding from the last walkthrough step. The walkthrough encoding is passed as

an input to the LSTM in a recurrent fashion.

6. Experiments

This section provides the results for several baseline approaches that achieve state-of-the-art performance on other embodied tasks (*e.g.* navigation). The room rearrangement task and the RoomR dataset are very challenging. To make the problem more manageable, we simplify assumptions in choosing the action space and the sensor modalities. Sec. 6.1 and Sec. 6.2 explain the details of the action space and sensor modalities, respectively. We show that even with these simplifications, the baseline models struggle.

6.1. Action Space

AI2-THOR offers a wide variety of means by which agents may interact with their environment ranging from “low-level” (*e.g.* applying forces to individual objects) to “high-level” (*e.g.* open an object of the given type) interactions. Prior work, *e.g.* [24, 45, 23, 18, 41] has primarily used higher-level actions to abstract away some details that would otherwise distract from the problem of interest. We follow this prior work and define our agent’s action space as $\mathcal{A} = \mathcal{A}_{\text{Nav}} \cup \mathcal{A}_{\text{Rotate}} \cup \mathcal{A}_{\text{Look}} \cup \mathcal{A}_{\text{UpDown}} \cup \mathcal{A}_{\text{Pickup}} \cup \mathcal{A}_{\text{Open}} \cup \{\text{PLACEOBJECT}, \text{DONE}\}$ where taking action:

- $a \in \mathcal{A}_{\text{Nav}} = \{\text{MOVE}X \mid X \in \{\text{AHEAD}, \text{LEFT}, \text{RIGHT}, \text{BACK}\}\}$ results in the agent moving 0.25m in the direction specified by X in the agent’s coordinate frame (unless this would result in the agent colliding with an object).
- $a \in \mathcal{A}_{\text{Rotate}} = \{\text{ROTATELEFT}, \text{ROTATERIGHT}\}$ results in the agent rotating 90° clockwise if $a = \text{ROTATERIGHT}$ and 90° counter-clockwise if $a = \text{ROTATELEFT}$.
- $a \in \mathcal{A}_{\text{Look}} = \{\text{LOOKUP}, \text{LOOKDOWN}\}$ results in the agent lowering/raising its camera angle by 30°,
- $a \in \mathcal{A}_{\text{Pickup}} = \{\text{PICKUP}X \mid X \in \{\text{the 62 pickupable object types}\}\}$ results in the agent picking up a visible object of type X if: (a) the agent is not already holding an object, (b) the agent is close enough to the object (within 1.5m), and (c) picking up the object would not result in it colliding with objects in front of the agent. If there are multiple objects of type X then the closest is chosen.
- $a \in \mathcal{A}_{\text{UpDown}} = \{\text{STAND}, \text{CROUCH}\}$ results in the agent raising or lowering the agent’s camera to one of two fixed heights allowing it to, *e.g.*, see objects under tables.
- $a \in \mathcal{A}_{\text{Open}} = \{\text{OPEN}X \mid X \in \{\text{the 10 openable object types that are not pickupable}\}\}$, if an object whose openness is different from the openness in the goal state is visible and within 1.5m of the agent, this object’s openness is changed to its value in the goal state.
- $a = \text{PLACEOBJECT}$ results in the agent dropping its held object. If the held object’s goal state is visible and within

1.5m of the agent, it is placed into that goal state. Otherwise, a heuristic is used to place the object on a nearby surface.

- $a = \text{DONE}$ results in the walkthrough or unshuffle stage immediately terminating.

In total, there are $|\mathcal{A}| = 84$ possible actions. Some of the above actions have been designed to be fairly abstract or “high-level,” *e.g.* the PLACEOBJECT action abstracts away all object manipulation complexities. As we discuss in Appendix C, we have implemented “lower-level” actions. Still, we stress that, even with these more abstract actions, the planning and visual reasoning required in RoomR already makes the task very challenging.

6.2. RoomR Variants

We will now detail the *1-Phase* and, more difficult, *2-Phase* variants of our RoomR task. These variants are, in part, defined by the sensors available to the agent. We begin by listing all sensors (note that only a subset of these will be available to any given agent in the below variants):

- **RGB** – An egocentric 224×224×3 RGB image corresponding to the agent’s current viewpoint (90° FOV). In the *1-Phase* task this corresponds to the RGB image from the unshuffle stage.
- **WALKTHROUGHRGB** – This sensor is only available in the *1-Phase* task and is identical to RGB except it shows the egocentric image as though the agent was in the Walkthrough stage, *i.e.* all objects were in their goal positions. It is this sensor that makes it possible, during the *1-Phase* task, for the agent to perform pixel-to-pixel comparisons between the environment as it should be in the walkthrough stage and as it is during the unshuffle stage.
- **AGENTPOSITION** – The agent’s position relative to its starting location (this is equivalent to the assumption of perfect egomotion estimation).
- **INWALKTHROUGH** – Only relevant during the *2-Phase* task, this sensor returns “true” if the agent is currently in the walkthrough stage and otherwise returns “false”.

1-Phase Task – In this variant, the agent takes actions within the walkthrough and unshuffle stages simultaneously in lock step. That is, if the agent takes a MOVEAHEAD action, the agent moves ahead in both stages simultaneously; as the agent begins in the same starting position in both stages, the agent’s position will always be the same in both stages. As only navigational actions are allowed during walkthrough, all actions of type $\mathcal{A}_{\text{Pickup}} \cup \mathcal{A}_{\text{Open}} \cup \{\text{PLACEOBJECT}\}$ are not executed by the agent in the walkthrough stage. During the unshuffle stage, the agent has access to the RGB, WALKTHROUGHRGB, and AGENTPOSITION sensors to complete its task.

2-Phase Task – In this task, the agent must complete both the walkthrough and unshuffle stages sequentially. In this

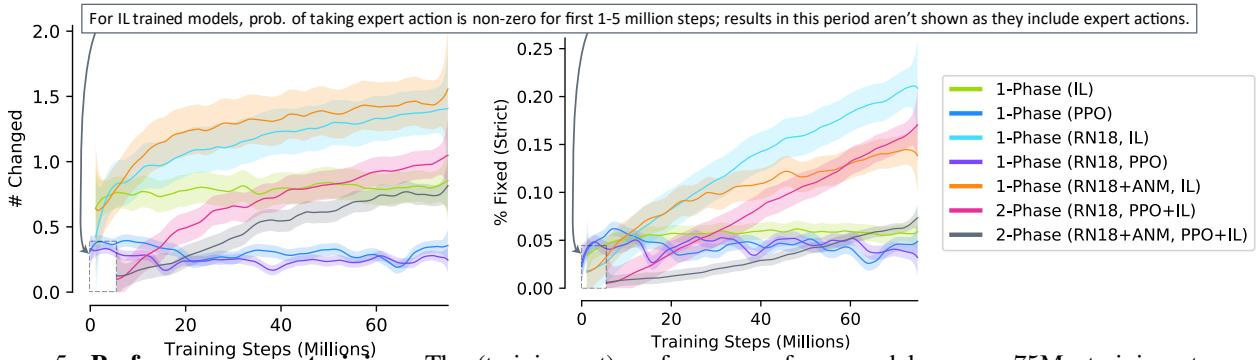


Figure 5: Performance over training. The (training-set) performance of our models over $\sim 75\text{Mn}$ training steps. We report the `#CHANGED` and `%FIXEDSTRICT` metrics, shown values and 95% error bars are generated using locally weighted scatterplot smoothing. Notice that the PPO models quickly saturate suggesting that they become stuck in local optima. IL continue to improve throughout training although Tab. 1 suggests that these models begin to overfit on the training scenes.

Experiment	100 · SUCCESS \uparrow			100 · %FIXEDSTRICT \uparrow			%E \downarrow			#CHANGED \ddagger		
	Train	Val.	Test	Train	Val.	Test	Train	Val.	Test	Train	Val.	Test
1-Phase (Simple, IL)	2.2	1.3	1.8	7.3	4.7	4.8	1.17	1.10	1.08	1.1	0.7	0.6
1-Phase (Simple, PPO)	1.8	2.1	0.7	6.7	6.7	4.6	0.95	0.96	0.99	0.3	0.4	0.4
1-Phase (RN18, IL)	8.2	1.7	2.8	17.9	5.0	6.3	0.93	1.14	1.11	1.3	0.9	0.9
1-Phase (RN18, PPO)	1.4	1.5	1.1	6.6	6.0	5.3	0.94	0.96	0.98	0.3	0.3	0.3
1-Phase (RN18+ANM, IL)	4.8	5.2	3.2	12.8	11.1	8.9	1.05	1.05	1.04	1.3	1.0	1.0
2-Phase (RN18, PPO+IL)	1.6	0.5	0.2	4.2	1.2	0.7	1.10	1.15	1.12	0.6	0.4	0.4
2-Phase (RN18+ANM, PPO+IL)	2.3	0.6	0.3	7.3	1.6	1.4	1.09	1.15	1.10	0.9	0.5	0.4
Heur. Expert	85.1	88.0	83.4	91.2	93.1	91.2	0.09	0.07	0.09	2.2	2.2	2.3

Table 1: Results. For each experiment, (i) we evaluate model checkpoints, saved after approximately 0, 5, ..., 75 million steps, on the validation set, (ii) choose the best performing (lowest avg. `%FIXEDSTRICT`) checkpoint among these, and (iii) evaluate this best validation checkpoint on the other dataset splits. \uparrow and \downarrow denote if larger or smaller metric values are to be preferred, \ddagger denotes a metric that is meant to highlight behavior rather than a measure quality.

task has access to the RGB, AGENTPOSITION, and IN-WALKTHROUGH sensors.

6.3. Training Pipeline

As our experimental results show, we found training models to complete the RoomR task using purely reward-based reinforcement learning methods to be extremely challenging. The difficulty remains even when using dense, shaped rewards. Thus, we have chosen to adopt a hybrid training strategy where we use the DD-PPO [47, 40] algorithm, a reward-based RL method, to train our agent when it is within the walkthrough stage, and an imitation learning (IL) approach, where we minimize a cross-entropy loss between the agent’s policy and expert actions, is used when in the unshuffle stage. As it has been successfully employed in training agents in other embodied tasks (*e.g.* [20]), for our IL training, we employ DAgger [37]. In DAgger, we begin training by forcing our agent to always take an expert’s action with probability 1 and anneal this probability to 0 over the first 1Mn for the *1-Phase* task and 5Mn steps for the *2-Phase* task. Tacitly assumed in the above is that we have access to an expert policy which can be efficiently evaluated at every state reached by our agent. Even with access to the full environment state, hand-designing an optimal,

efficiently computable, expert is extremely difficult: simple considerations show that planning the agent’s route is at least as difficult as the traveling salesman problem. Therefore, we do not attempt to design an optimal expert and, instead, a greedy heuristic expert with some backtracking and error detection capabilities. See Appendix B for more details. This expert is not perfect but, as seen in Tab. 1, can restore all but a small fraction of objects to their rightful places. For additional training details, see Appendix A.

6.4. Results

Recall from Sec. 4 that our dataset contains a training set of size 4000 and validation/testing sets of 1000 instances each. We report results on each of these splits but, for efficiency, include only the first 15 rearrangement instances per room in the training set (leaving 1200 instances).

Baselines. We evaluate the following baseline models:

- *1-Phase (RN18, IL)* – An agent trained using pure imitation learning in the *1-Phase* task. Recall that *1-Phase* task models use a simplification of the model from Sec. 5, see our code for more details.
- *1-Phase (RN18, PPO)* – As above but trained with PPO.
- *1-Phase (Simple, IL)* – As *1-Phase (RN18, IL)* but we replace the ResNet18 CNN backbone and attention module

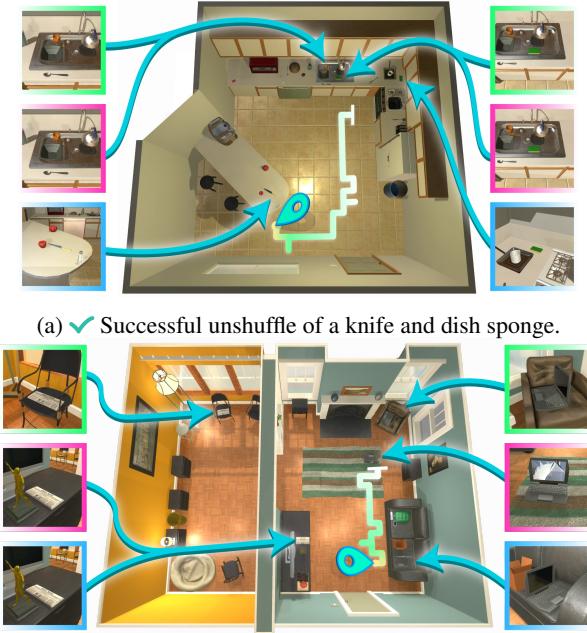


Figure 6: **Qualitative results.** Trajectories sampled from a *1-Phase* model. The goal, predicted, and initial configurations are green, pink, and blue, respectively.

with 3 CNN blocks, this CNN is commonly used in embodied navigation baselines [38].

- *1-Phase (Simple, PPO)* – As *PointU (Simple, IL)* but trained with PPO rather than IL.
- *2-Phase (RN18, PPO+IL)* – An agent trained in the *2-Phase* task using the model from Sec. 5. PPO and IL are used in the walkthrough and unshuffle stages, respectively.
- *1-Phase (RN18+ANM, IL)* – We pretrain a variant of the “Active Neural SLAM” (ANM) [8] architecture to perform semantic mapping within AI2-THOR using our set of 72 object categories. We then freeze this mapping network and train our “*1-Phase (RN18, IL)*” model extended to allow for comparing between the maps created in the unshuffle and walkthrough stages. See Appendix D for more details.
- *2-Phase (RN18+ANM, PPO+IL)* – Similarly as above but with semantic mapping model integrated into “*2-Phase (RN18, PPO+IL)*” baseline above.

Analysis. We record rolling metrics during training in Fig. 5. After training, we evaluate our models on our three dataset splits and record the average metric values in Tab. 1. From the results, we see several clear trends.

Unshuffling objects is hard – Even when evaluated on the seen training rearrangements in the easier *1-Phase* task, the success of our best model is only 8.2%.

- *Reward-based RL struggles to train* – Fig. 5 shows that PPO-based models quickly appear to become trapped in local optima. Tab. 1 shows that the PPO agents move relatively few objects but, when they do move objects, they

generally place them correctly even in test scenes.

- *Pretrained CNN backbones can improve performance* – We hypothesized that using a pretrained CNN backbone would substantially improve generalization performance given the relatively little object variety (compared with ImageNet) in our dataset. We see compelling evidence of this when comparing the performance of the “*1-Phase (Simple, IL)*” and “*1-Phase (RN18, IL)*” baselines (SUCCESS and %FIXEDSTRICT improvements across all splits). The results were more mixed for the PPO-trained baselines.

• *The 2-Phase task is much more difficult than the 1-Phase task* – Comparing the performance of the “*2-Phase (RN18, PPO+IL)*” and “*1-Phase (RN18, IL)*” baselines, it is clear that the *2-Phase* task is much more difficult than the *1-Phase* task. If the agent managed to explore exhaustively during the walkthrough stage then the two tasks would be effectively identical. This suggests that the observed gap is primarily driven by learning dynamics and the walkthrough agent’s failure to explore exhaustively. Note that, as we select the best val. set model, Tab. 1 may give the impression that the *2-Phase* baseline failed to train at all: this is not the case as we can see, in Fig. 5, that the “*2-Phase (RN18, PPO+IL)*” baseline trains to almost the same training-set performance as the *1-Phase* IL baselines.

- *Semantic mapping appears to substantially improve performance* – Our preliminary results suggest that semantic mapping can have a substantial impact on improving the generalization performance of rearrangement models, note that the “+ANM” baselines outperform their counterparts in almost all metrics, especially so on the validation and test sets. These results are preliminary as we have not carefully balanced parameter counts to ensure fair comparisons.

See Fig. 6 for success and failure examples.

7. Discussion

Our proposed Room Rearrangement task poses a rich set of challenges, including navigation, planning, and reasoning about object poses and states. To facilitate learning for rearrangement, we propose the RoomR dataset that provides a challenging testbed in visually rich interactive environments. We show that modern deep RL methodologies obtain (test-set) performance only marginally above chance. Given the low performance of existing methods we suspect that future high-performance models will require novel architectures enabling comparative mapping (to record object positions during the walkthrough stage and compare these positions against those observed in the unshuffle stage), visual reasoning about object positions, and physics to be able to manipulate objects to their goal locations. Moreover, we require new reinforcement learning methodologies to allow the walkthrough and unshuffle stages to be trained jointly with minimum mutual interference. Given these challenges, we hope the proposed task opens up new avenues of research in the domain of Embodied AI.

References

- [1] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. On evaluation of embodied navigation agents. *arXiv*, 2018. [1](#), [2](#)
- [2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *CVPR*, 2018. [1](#), [2](#)
- [3] Dhruv Batra, Angel X. Chang, Sonia Chernova, Andrew J. Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, Manolis Savva, and Hao Su. Rearrangement: A challenge for embodied ai. *arXiv*, 2020. [2](#)
- [4] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, A. Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv*, 2020. [1](#), [2](#)
- [5] Ohad Ben-Shahar and Ehud Rivlin. Practical pushing planning for rearrangement tasks. *ICRA*, 1996. [2](#)
- [6] Lars Berscheid, Pascal Meißner, and Torsten Kröger. Self-supervised learning for precise pick-and-place without object model. *IEEE Robotics and Automation Letters*, 2020. [2](#)
- [7] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *NeurIPS*, 2020. [2](#)
- [8] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *ICLR*, 2020. [2](#), [8](#), [12](#)
- [9] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. In *ICLR*, 2019. [2](#)
- [10] Nikolaus Correll, Kostas E. Bekris, Dmitry Berenson, Oliver Brock, Albert Causo, Kris Hauser, Kei Okada, Alberto Rodriguez, Joseph M. Romano, and Peter R. Wurman. Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 2018. [2](#)
- [11] Akansel Cosgun, Tucker Hermans, Victor Emeli, and Mike Stilman. Push planning for object placement on cluttered table surfaces. In *IROS*, 2011. [2](#)
- [12] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *RSS*, 2016. [2](#)
- [13] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018. [2](#)
- [14] 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](#)
- [15] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. In *RSS*, 2011. [2](#)
- [16] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anshit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *CORL*, 2018. [2](#)
- [17] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Ffrob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research*, 2018. [2](#)
- [18] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. IQA: Visual question answering in interactive environments. In *CVPR*, 2018. [2](#), [6](#)
- [19] Marcus Gualtieri, Andreas ten Pas, and Robert Platt. Pick and place without geometric object models. In *ICRA*, 2018. [2](#)
- [20] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *Int. J. Comput. Vis.*, 128(5):1311–1330, 2020. [7](#)
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. [5](#)
- [22] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997. [5](#)
- [23] Unnat Jain, Luca Weihs, Eric Kolve, Ali Farhadi, Svetlana Lazebnik, Aniruddha Kembhavi, and Alexander G. Schwing. A cordial sync: Going beyond marginal policies for multi-agent embodied tasks. In *ECCV*, 2020. [2](#), [5](#), [6](#)
- [24] 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. [2](#), [6](#)
- [25] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, 2011. [2](#)
- [26] Jennifer King, Joshua Haustein, Siddhartha Srinivasa, and Tamim Asfour. Nonprehensile whole arm rearrangement planning on physics manifolds. In *ICRA*, 2015. [2](#)
- [27] Jennifer E King, Marco Cognetti, and Siddhartha S Srinivasa. Rearrangement planning using object-centric and robot-centric action spaces. In *ICRA*, 2016. [2](#)
- [28] D. Kingma and J. Ba. A method for stochastic optimization. In *CVPR*, 2017. [11](#)
- [29] 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. [2](#), [3](#)
- [30] Athanasios Krontiris and Kostas E Bekris. Dealing with difficult instances of object rearrangement. In *RSS*, 2015. [2](#)
- [31] A. Krontiris and K. E. Bekris. Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In *ICRA*, 2016. [2](#)
- [32] Athanasios Krontiris, Rahul Shome, Andrew Dobson, Andrew Kimmel, and Kostas Bekris. Rearranging similar objects with a manipulator using pebble graphs. In *IEEE-RAS International Conference on Humanoid Robots*, 2014. [2](#)

- [33] Yann Labb  , Sergey Zagoruyko, Igor Kalevatykh, Ivan Laptev, Justin Carpentier, Mathieu Aubry, and Josef Sivic. Monte-carlo tree search for efficient visually guided rearrangement planning. *IEEE Robotics and Automation Letters*, 2020. 2
- [34] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016. 5
- [35] Erion Plaku and Gregory D Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *ICRA*, 2010. 2
- [36] Santhosh K. Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. In *ECCV*, 2020. 2
- [37] St  phane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011. 2, 7
- [38] 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. In *ICCV*, 2019. 2, 5, 8
- [39] Jonathan Scholz and Mike Stilman. Combining motion planning and optimization for flexible robot manipulation. In *IEEE-RAS International Conference on Humanoid Robots*, 2010. 2
- [40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017. 2, 7, 11
- [41] 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. 1, 2, 6
- [42] Haoran Song, Joshua A Haustein, Weihao Yuan, Kaiyu Hang, Michael Yu Wang, Danica Kragic, and Johannes A Stork. Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting. In *IROS*, 2020. 2
- [43] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *ICRA*, 2014. 2
- [44] Mike Stilman, Jan-Ullrich Schamburek, James Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *ICRA*, 2007. 2
- [45] Luca Weihs, Aniruddha Kembhavi, Kiana Ehsani, Sarah M. Pratt, Winson Han, Alvaro Herrasti, Eric Kolve, Dustin Schwenk, Roozbeh Mottaghi, and Ali Farhadi. Learning generalizable visual representations via interactive gameplay. In *ICLR*, 2021. 5, 6
- [46] 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. 11
- [47] 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. 2, 5, 7
- [48] 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. 2
- [49] Fei Xia, Chengshu Li, Roberto Mart  n-Mart  n, Or Litany, Alexander Toshev, and Silvio Savarese. Relmogen: Leveraging motion generation in reinforcement learning for mobile manipulation. *arXiv*, 2020. 1
- [50] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. Sapien: A simulated part-based interactive environment. In *CVPR*, 2020. 1
- [51] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. In *ICLR*, 2019. 2
- [52] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CoRL*, 2020. 2
- [53] Weihao Yuan, Johannes A Stork, Danica Kragic, Michael Y Wang, and Kaiyu Hang. Rearrangement with nonprehensile manipulation using deep reinforcement learning. In *ICRA*, 2018. 2
- [54] Kevin Zakka, Andy Zeng, Johnny Lee, and Shuran Song. Form2fit: Learning shape priors for generalizable assembly from disassembly. In *ICRA*, 2020. 2
- [55] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual semantic planning using deep successor representations. In *ICCV*, 2017. 2

Appendix

A. Implementation details

We train our agents using the AllenAct Embodied AI framework [46] for ~ 75 Mn steps. We run our experiments on g4dn.12xlarge Amazon EC2 instances which has 4 NVIDIA T4 GPUs and 48 CPU cores. See Table 2 for an accounting of our training hyperparameters (e.g. learning rate, loss weights, etc.). During training we obtain an FPS of ~ 125 when training models with expert supervision and an FPS of ~ 300 when training purely with PPO. Thus, training for ~ 75 Mn steps requires approximately 4.3 days when using expert supervision and 1.8 without.

The reward structures for our agents differ in the walkthrough and unshuffle stages. Rather than provide explicit details here, as these are better read directly from code, we give some intuition about these reward structures.

Unshuffle stage rewards. For the unshuffle stage the reward is quite simple. Suppose that before the agent takes an action the scene is in state $s^1 \in \mathcal{S}$ and, after the agent takes a step, the scene is in state $s^2 \in \mathcal{S}$. The agent’s reward is then equal to the change in energy of the scene (with respect to the goal pose s^*), i.e. $D(s^1, s^*) - D(s^2, s^*)$. Thus if the energy has decreased ($D(s^2, s^*) < D(s^1, s^*)$) so that the scene is closer to the goal state than it was before, then the agent gets a positive reward. Otherwise, the agent may receive a negative reward. At the end of an unshuffle episode the agent receives a penalty equal to the negation of the remaining energy.

Walkthrough stage rewards. In the walkthrough stage we would like the agent to see as many of the objects in the scene as possible so that, during the unshuffle stage, the agent can compare the object poses seen against their goal positions. To this end, after every step in the walkthrough stage, the agent receives reward if it observes objects that it has never seen previously in the episode. At the end of the episode we provide the agent a reward based on the proportion of objects the agent has seen among all objects in the scene. We found this reward helpful to encourage the agent to be as exhaustive as possible.

B. Heuristic Expert

The sole purpose of our expert is to produce expert actions for our learning agents to imitate. As such it is allowed to “cheat” by using extensive ground truth state information including the scene layout and poses of all objects in current and goal states. As it does not have to reason from visual input, the heuristic expert’s performance cannot be fairly compared against the other agents. At a high-level our expert operates by looping through (1) selecting the closest object that is not in its goal pose, (2) navigating to this object via shortest paths computed on the scene layout, (3)

Hyperparameter	Value
<i>PPO</i>	
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Value loss coefficient	0.5
Entropy loss coefficient	0.01
Clip parameter (ϵ) [40]	0.1
Decay on ϵ	Linear(1, 0.39, 75e6)
<i>PPO-only – Training</i>	
# Processes to sample steps	40 (5 per GPU)
LR Decay	Linear(1, 1/3, 25e6)
<i>IL and IL+PPO – Training</i>	
# Processes to sample steps	40 (5 per GPU)
<i>Common – Training</i>	
Rollout timesteps	64
Rollouts per minibatch	40
Epochs	3
Learning rate	3e-4
Optimizer	Adam [28]
(β_1, β_2) for Adam	(0.9, 0.999)
Gradient clip norm	0.5
Training steps	75 Million

Table 2: **Training hyperparameters.** Here $\text{Linear}(a, b, c)$ corresponds to linear interpolation between a and b within c training steps.

picking up the object, (4) navigating to the closest position from which the object can be placed in its goal pose, and (5) placing the object. As AI2-THOR is physics based, it is possible for the above steps to fail (e.g. an object falls in the way of the agent as it navigates), because of this the agent has backtracking capabilities to allow it to give up on placing an object temporarily in the hope that, in placing other objects, it will remove the obstruction.

C. Lower-level actions

As discussed in Sec. 6.1, in our experiments we use a “high-level” action space in line with prior work. We suspect (and hope) that within the next few years the rearrangement task will be solved using these high-level actions enabling us to move to low-level actions which are more easily implementable on existing robotic hardware. In preparation for this eventuality, we have implemented a number of lower-level actions. Rather than describe these actions individually, we will describe them in contrast to their higher-level counterparts.

Continuous navigation. In our experiments the agent moves at increments of 0.25 meters, uses 90° rotations, and changes its camera angle by 30° at a time. We have implemented fully continuous motion so that the agent can rotate

and move arbitrary degrees and distances respectively.

Object manipulation. Our high-level actions include a PLACEOBJECT action that abstracts away the subtleties of moving a held object to a goal location. In our low-level actions we now allow the agent to move a held object through space (within some distance of the agent) possibly colliding with other objects. The agent then must explicitly drop the object into to the goal location.

Opening and picking up objects. When an agent opens an object using one of the 10 high-level open actions the agent is not required to specify the target openness nor specify where, in space, the object to open resides. Fig. 7 shows how objects are targeted with our lower-level actions. For our low-level open action the agent must specify the (x, y) coordinates (in pixel-space) of the object, as well as the amount that the object is opened. Similarly, our low level PICKUP action requires specifying the object with (x, y) coordinates rather than by the object’s type.

D. Semantic Mapping

As discussed in the main paper, we include two baselines that incorporate the “Active Neural SLAM” module of Chaplot et al. (2020) [8] which we have adapted (by in-

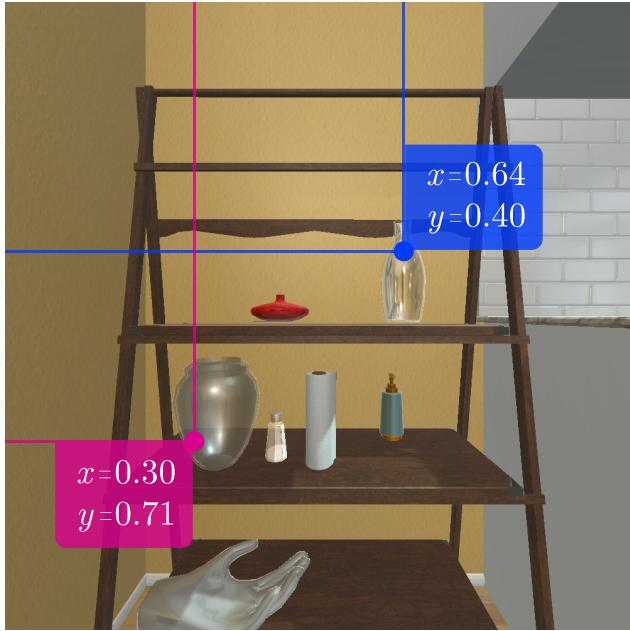


Figure 7: **Lower-level object targeting.** Instead of targeting objects based on their annotated type, the lower-level targeting action targets objects based on their location in the agent’s current frame. For each (x, y) coordinate, with $0 \leq x, y \leq 1$, the x and y coordinates denote the relative distance from the left and top of the frame, respectively.

creasing the number of output channels in the map) to perform semantic mapping.

We pretrain the ANM module so that, given a $224 \times 224 \times 3$ image from AI2-THOR, it returns a $40 \times 40 \times 75$ tensor M corresponding to an estimate of the semantic map in a $2m \times 2m$ region directly in front of the agent (3 channels are used to predict free space, the other 72 are used to predict the probability that one of our 72 rearrangement objects occupies a given map location).

After pretraining this module we freeze its weights and incorporate it into our baseline model, recall Sec. 5. In particular, we remove the nonparametric map from our baseline and replace it with the ANM. During the walkthrough stage the agent constructs the semantic map and saves it. During the unshuffle stage, the agent indexes into the walkthrough map to retrieve the estimate of the egocentric semantic map for the agent’s current position. It compares this walkthrough map estimate against its current map estimate through the use of an attention mechanism: the two estimates are concatenated, embedded via a CNN, and then attention is computed spatially to downsample the embeddings to a single 512-dimensional vector. This embedding is then concatenated to the input to the 1-layer LSTM (recall Sec. 5) along with the usual visual and discrete embeddings.

E. Computing the energy between two poses

In our discussion of the “% Energy Remaining” metric (recall Sec. 3.2) we deferred the definition of the energy function $D : S \times S \rightarrow [0, 1]$, we define this energy function now. Let $s^i = (p^i, o^i, c^i, b^i), s^2 = (p^2, o^2, c^2, b^2) \in S$ be two possible poses for an object. Then,

- If $b^1 = 1$ or $b^2 = 1$ we let $D(s^1, s^2) = 1$.
- Otherwise, if the object is openable but not pickupable, we let $D(s^1, s^2) = 0$ if $|o^1 - o^2| \leq 0.2$ and otherwise $D(s^1, s^2) = 1$, otherwise
- Otherwise, if the object is pickupable, we have two cases. Suppose that $\text{IOU}(s^1, s^2) > 0$. Then we let $D(s^1, s^2) = 0.5 \cdot \max(0, 0.5 - \text{IOU}(s^1, s^2))$. Otherwise, we let $D(s^1, s^2) = 0.5 + 0.5 \cdot \min(d/2, 1)$ where d be the minimum distance between a point in c^1 and a point in c^2 .

Note that D decreases monotonically as poses p^1, p^2 come closer together.

F. Object types

The list of all objects have been provided in Tab. 3.

Object Type [A-L]	Openable	Pickupable	Object Type [M-Z]	Openable	Pickupable
AlarmClock	✗	✓	Microwave	✓	✗
AluminumFoil	✗	✓	Mirror	✗	✗
Apple	✗	✓	Mug	✗	✓
ArmChair	✗	✗	Newspaper	✗	✓
BaseballBat	✗	✓	Ottoman	✗	✗
BasketBall	✗	✓	Painting	✗	✗
Bathtub	✗	✗	Pan	✗	✓
BathtubBasin	✗	✗	PaperTowelRoll	✗	✓
Bed	✗	✗	Pen	✗	✓
Blinds	✓	✗	Pencil	✗	✓
Book	✓	✓	PepperShaker	✗	✓
Boots	✗	✓	Pillow	✗	✓
Bottle	✗	✓	Plate	✗	✓
Bowl	✗	✓	Plunger	✗	✓
Box	✓	✓	Poster	✗	✗
Bread	✗	✓	Pot	✗	✓
ButterKnife	✗	✓	Potato	✗	✓
CD	✗	✓	RemoteControl	✗	✓
Cabinet	✓	✗	RoomDecor	✗	✗
Candle	✗	✓	Safe	✓	✗
CellPhone	✗	✓	SaltShaker	✗	✓
Chair	✗	✗	ScrubBrush	✗	✓
Cloth	✗	✓	Shelf	✗	✗
CoffeeMachine	✗	✗	ShelvingUnit	✗	✗
CoffeeTable	✗	✗	ShowerCurtain	✓	✗
CounterTop	✗	✗	ShowerDoor	✓	✗
CreditCard	✗	✓	ShowerGlass	✗	✗
Cup	✗	✓	ShowerHead	✗	✗
Curtains	✗	✗	SideTable	✗	✗
Desk	✗	✗	Sink	✗	✗
DeskLamp	✗	✗	SinkBasin	✗	✗
Desktop	✗	✗	SoapBar	✗	✓
DiningTable	✗	✗	SoapBottle	✗	✓
DishSponge	✗	✓	Sofa	✗	✗
DogBed	✗	✗	Spatula	✗	✓
Drawer	✓	✗	Spoon	✗	✓
Dresser	✗	✗	SprayBottle	✗	✓
Dumbbell	✗	✓	Statue	✗	✓
Egg	✗	✓	Stool	✗	✗
Faucet	✗	✗	StoveBurner	✗	✗
Floor	✗	✗	StoveKnob	✗	✗
FloorLamp	✗	✗	TVStand	✗	✗
Footstool	✗	✓	TableTopDecor	✗	✓
Fork	✗	✓	TeddyBear	✗	✓
Fridge	✓	✗	Television	✗	✗
GarbageBag	✗	✗	TennisRacket	✗	✓
GarbageCan	✗	✗	TissueBox	✗	✓
HandTowel	✗	✓	Toaster	✗	✗
HandTowelHolder	✗	✗	Toilet	✓	✗
HousePlant	✗	✗	ToiletPaper	✗	✓
Kettle	✓	✓	ToiletPaperHanger	✗	✗
KeyChain	✗	✓	Tomato	✗	✓
Knife	✗	✓	Towel	✗	✓
Ladle	✗	✓	TowelHolder	✗	✗
Laptop	✓	✓	VacuumCleaner	✗	✗
LaundryHamper	✓	✗	Vase	✗	✓
Lettuce	✗	✓	Watch	✗	✓
LightSwitch	✗	✗	WateringCan	✗	✓
			Window	✗	✗
			WineBottle	✗	✓

Table 3: **Object types.** All object types available in AI2-THOR (and thus present in our task) along with whether they are openable or pickupable.