Visual Room Rearrangement

Luca Weihs\textsuperscript{1}  Matt Deitke\textsuperscript{1,2}  Aniruddha Kembhavi\textsuperscript{1,2}  Roozbeh Mottaghi\textsuperscript{1,2} \\
\textsuperscript{1}PRIOR @ Allen Institute for AI  \textsuperscript{2}University of Washington \\
ai2thor.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

---

1https://youtu.be/1AFxaOC9U-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

- \( p_i \in \{ \text{3D rotations and translations} \} = \text{SE}(3) \) 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 \( o_i = 0 \),
- \( c_i \) 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 \( S = \text{SE}(3) \times ([0, 1] \cup \{ 0 \}) \times \mathbb{R}^{8 \times 3} \times \{ 0, 1 \} \) be the set of all possible poses for a single object and \( S = \prod_{i=1}^n S \) the set of all possible joint object poses. The agent’s goal is to convert an initial configuration \( s^0 \in S \) to a goal state \( s^* \in S \).

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 \( s^* \) (\( s^0 \to s^* \)) 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 \( s = (s_1, \ldots, s_n) \in S \). 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 object \( i \) 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)** (\%FIXEDS) – 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 \( \%\text{FIXEDS} \) 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 \( \%\text{FIXEDS} \) equal \( 1 - \frac{|M_{\text{end}}|}{|M_{\text{start}}|} \) (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 \( D : S \times S \to [0, 1] \) 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.

\[
\%E = \frac{\left( \sum_{i=1}^n D(s_i, s_i^*) \right)}{\left( \sum_{i=1}^n D(s_i^0, s_i^*) \right)}.
\]

**# Changed** (#CHANGED) – To give additional insight as to the 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
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 $0.25m \times 0.25m$. 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 $M \in \{1 - N, \ldots, 5 - N\}$. 

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

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-
with frozen model weights with the final average pooling and classification layers removed. This ResNet18 model transforms input images into 7×7×512 tensors. For our RNN, we leverage a 1-layer LSTM [22] with 512 hidden units. To produce the policies \( \pi_{\text{walk}} \) 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 \( v_{\text{walk}} \) 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

![Figure 4: Model overview.](image)

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.
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 \( \mathcal{A} = \mathcal{A}_{\text{Nav}} \cup \mathcal{A}_{\text{Rotate}} \cup \mathcal{A}_{\text{Look}} \cup \mathcal{A}_{\text{UpDown}} \cup \mathcal{A}_{\text{Pickup}} \cup \mathcal{A}_{\text{Open}} \cup \{\text{PLACEOBJECT, DONE}\} \) where taking action:

- \( a \in \mathcal{A}_{\text{Nav}} = \{\text{MOVE} | X \in \{\text{AHEAD, LEFT, RIGHT, BACK}\}\} \) 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).
- \( a \in \mathcal{A}_{\text{Rotate}} = \{\text{ROTATELEFT, ROTATERIGHT}\} \) results in the agent rotating 90° clockwise if \( a = \text{ROTATERIGHT} \) and 90° counter-clockwise if \( a = \text{ROTATELEFT} \).
- \( a \in \mathcal{A}_{\text{Look}} = \{\text{LOOKUP, LOOKDOWN}\} \) results in the agent lowering/raising its camera angle by 30°.
- \( a \in \mathcal{A}_{\text{Pickup}} = \{\text{PICKUPX} | X \in \{\text{the 62 pickupable object types}\}\} \) 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.
- \( a \in \mathcal{A}_{\text{UpDown}} = \{\text{STAND, CROUCH}\} \) 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.
- \( a \in \mathcal{A}_{\text{Open}} = \{\text{OPENX} | X \in \{\text{the 10 openable object types that are not pickupable}\}\} \), 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 \( \text{PLACEOBJECT} \) 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
suggests that these models begin to overfit on the training scenes. 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.

![Figure 5: Performance over training. The (training-set) performance of our models over ~75Mn training steps. We report the \#CHANGED and %FIXED\text{STRICT} 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.](image)

<table>
<thead>
<tr>
<th>Experiment</th>
<th>100 - SUCCESS †↑</th>
<th>100 - %FIXED\text{STRICT} †↑</th>
<th>%E ‡↓</th>
<th>#CHANGED ‡↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train Val.</td>
<td>Train Val. Test</td>
<td>Train Val. Test</td>
<td>Train Val. Test</td>
<td>Train Val. Test</td>
</tr>
<tr>
<td>1-Phase (Simple, IL)</td>
<td>2.2 1.3 1.8</td>
<td>7.3 4.7 4.8</td>
<td>1.17 1.10 1.08</td>
<td>1.1 0.7 0.6</td>
</tr>
<tr>
<td>1-Phase (Simple, PPO)</td>
<td>1.8 2.1 0.7</td>
<td>6.7 6.7 4.6</td>
<td>0.95 0.96 0.99</td>
<td>0.3 0.4 0.4</td>
</tr>
<tr>
<td>1-Phase (RN18, IL)</td>
<td>8.2 1.7 2.8</td>
<td>17.9 5.0 6.3</td>
<td>0.93 1.14 1.11</td>
<td>1.3 0.9 0.9</td>
</tr>
<tr>
<td>1-Phase (RN18, PPO)</td>
<td>1.4 1.5 1.1</td>
<td>6.6 6.0 5.3</td>
<td>0.94 0.96 0.98</td>
<td>0.3 0.3 0.3</td>
</tr>
<tr>
<td>1-Phase (RN18+ANM, IL)</td>
<td>4.8 5.2 3.2</td>
<td>12.8 11.1 8.9</td>
<td>1.05 1.05 1.04</td>
<td>1.3 1.0 1.0</td>
</tr>
<tr>
<td>2-Phase (RN18, PPO+IL)</td>
<td>1.6 0.5 0.2</td>
<td>4.2 1.2 0.7</td>
<td>1.10 1.15 1.12</td>
<td>0.6 0.4 0.4</td>
</tr>
<tr>
<td>2-Phase (RN18+ANM, PPO+IL)</td>
<td>2.3 0.6 0.3</td>
<td>7.3 1.6 1.4</td>
<td>1.09 1.15 1.10</td>
<td>0.9 0.5 0.4</td>
</tr>
</tbody>
</table>

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. %FIXED\text{STRICT}) checkpoint among these, and (iii) evaluate this best validation checkpoint on the other dataset splits. † and ‡ denote if larger or smaller metric values are to be preferred. † denotes a metric that is meant to highlight behavior rather than a measure quality.

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. Tactily 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
Successful unshuffle of a knife and dish sponge for success and failure examples. Unsuccessful unshuffle of a newspaper and laptop. 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 %FIXED\%STRICT 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


