Appendix

A. Faster-RCNN

Here we mention the architecture, data, and training details for the Faster-RCNN network.

Architecture. We use the Faster-RCNN default architecture from Detectron2 [57].

Data. We use $\sim$10k images from train rooms in AI2-THOR with bounding box labels and $\sim$127k train images from LVIS [24]. We choose this number of THOR images to ensure diversity of views without repetition of frames that would appear very similar. In total the detector is trained on 1235 classes (accounting for the class overlap between LVIS and the $\sim$100 THOR categories).

Training. We train using the default Detectron2 3x schedule. We train on a machine with eight GeForce GTX TITAN X NVIDIA GPUs. The training takes $\sim$2 days.

Inference. At inference, we treat the Faster-RCNN module as a region proposal network. Hence, we only require its detections for extracting node and edge features. Our proposed algorithm does not use the predicted class labels as input to create node and edge feature representations.

B. Policy

Here we mention the architecture, data, and training details for the policy network.

Architecture. The policy consists of three convolutional layers and a GRU. It is an actor-critic style network. The input is an image $I \in \mathbb{R}^{224\times224\times3}$. Note, we do not feed in the action from the last timestep. The output of the conv. backbone is a volume, $v \in \mathbb{R}^{24\times24\times512}$, which is flattened and projected by a linear layer into a feature $x \in \mathbb{R}^{512}$. x is taken as input to the GRU module, which maintains a hidden state $h \in \mathbb{R}^{512}$ and outputs another feature vector $z \in \mathbb{R}^{512}$. An linear actor head takes z and projects it to give logits over the eight discrete actions. A second linear critic head takes z and projects it to give the critic score.

Data. We conduct training rollouts within random starting locations drawn from the 80 train rooms in AI2-THOR.

Training. We adopt the AllenAct [53] framework for training. Specifically we use the DD-PPO [44, 54] algorithm to train our network. We train on machines that have 48 CPU cores and four T4 NVIDIA GPUs. We train for 200 million steps, which takes $\sim$2 days. We use default AllenAct PPO settings, with rollout episode length of 150 steps. We employ sparse rewards, which are computed based on known simulation state at training. The agent receives positive reward of 0.1 if it visits a new position (agent orientation is disregarded) and reward of 0.4 if it sees a new object within a rollout. There is also a step penalty of -0.01 and a failed action penalty of -0.03.

C. Continuous Scene Representation

Architecture. We use a standard ResNet-18 architecture. We modify `conv1` to take five channel input (three channels for RGB, a forth channel for the first binary box mask, and a fifth channel for the second binary box mask). Hence our ResNet takes input $\in \mathbb{R}^{224\times224\times5}$. After the ResNet, we have an MLP bottleneck projection head, which takes in a feature $\in \mathbb{R}^{512}$ and outputs a feature $\in \mathbb{R}^{512}$. Architectural details for the network that extracts object correspondence features are the same.

Data. As stated in the paper we capture 20 random agent poses in 5 different configurations (random object placement and scene textures) in the 80 different train rooms. This leads to a train dataset of $\sim$600k relations. The rest of the dataset is composed of a near even split between validation and test relations, yielding a total dataset size of $\sim$900k relations. We provide some train statistics to give a better understanding of the dataset. Of the $\sim$600k relations, $\sim$60k are node relations, the rest are directed edge relations. In total there are $\sim$3k different object instances across the $\sim$100 AI2-THOR categories.

Training. We train on a machine with eight GeForce GTX TITAN X NVIDIA GPUs. Our learning leverages InfoNCE [50] loss and builds on the MoCo framework [11, 26]. We use a relatively small queue of size of 1024 for negatives. The InfoNCE temperature parameter is 0.07 and the momentum update coefficient is 0.999. We train in minibatches of 512, with initial learning rate of 0.1, a cosine decay schedule, and standard SGD w/ momentum optimizer. Our model take less than one day to converge.

Constants. We must set three thresholds in our method (1) to determine within trajectory matches, (2) to determine object correspondences matches, and (3) to determine if an object has moved. If (1) cosine similarity between two matched node features is greater than 0.5 within a trajectory, we consider the instances as a true match. If cosine similarity of matched object features between trajectories is greater than 0.4, we consider the objects to be a true match. Finally, if cosine similarity of the node features drops below 0.8 after nodes have been matched via object features between trajectories, we consider the object to be a candidate object that has moved.

D. Linear Probes

Here we provide more relevant details for our linear probes.

Data. We consider two tasks, [SUPPORT] and [SIBLING]. For [SUPPORT] we create a balanced dataset with $\sim$2k positive examples of an object on top of another object. By reversing the order of the boxes for the input, we get another $\sim$2k examples of an object under another object. Finally we create a third category of $\sim$2k examples of unre-

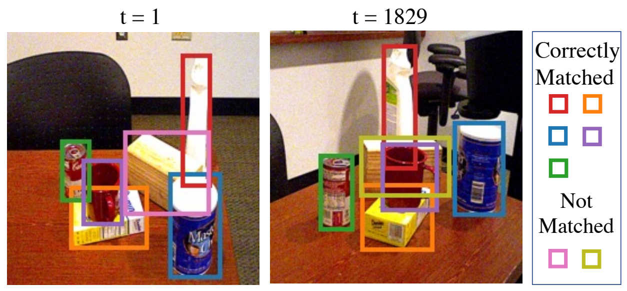

Figure 8. **Qualitative matching on YCB-Video.**. All but a heavily occluded.

lated objects (i.e., objects that do not follow the [SUPPORT] relationship). For the [SIBLING] relation we create two categories each with $\sim$2k examples, the first with examples from this relationship (i.e., two objects on the same receptacle) and the second of unrelated objects (i.e., objects that do not follow the [SIBLING] relationship). For both datasets we use a 80/20 train/test split for each category.

Training. For training our model, we conduct a linear probe, with the high learning rate of 0.5, using the validation set loss to determine convergence. We use features before the MLP projection head as is common for linear probes in the contrastive learning literature. For end-to-end baseline, we first train with reasonable parameters (i.e., learning rate of 0.02, with cosine schedule, SGD w/ momentum, weight decay of 0.001) for 100 epochs, taking the checkpoint with the lowest validation loss. We then use the same linear probe routine discussed above to probe the baseline representations for transfer performance.

E. Exploration Heuristic

To ablate the effectiveness of our exploration policy in the visual room rearrangement pipeline, we also design a heuristic policy. Given the simulation state of the room before and after the shuffle, we can retrieve the squares in the map that are closest (in terms of euclidean distance) to the objects that get shuffled. Hence, we get $2n$ locations, where n is the number of objects that get shuffled. During the walkthrough trajectory, based on the agent's current location, the heuristic policy greedily picks the closest location and takes the shortest path to this point. During the unshuffle exploration trajectory the locations are visited in reversed order (e.g., the waypoint visited last in the walkthrough is visited first in the unshuffle).

F. AI2-THOR Assets

AI2-THOR assets are available under Apache 2.0.

G. Rearrangement

While our setting is identical to that of Weihs *et al.* [52], our method does not attempt to fix objects that have changed state (e.g., drawers opening). Hence our method cannot successfully rearrange rooms with these changes. However, for fair comparison to prior work, we report numbers on the full RoomR dataset with all data points. Planning for objects that change in openness is left to future work.

H. Qualitative Real World Tracking Results

We show a qualitative example where all but one heavily occluded object is properly matched in Fig. 8.