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) 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.

Hyperparamter	**Value**
PPO	
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	$\texttt{Linear}(1, 0.39, 75e6)$
PPO-only – Training	
# Processes to sample steps	40 (5 per GPU)
LR Decay	$\texttt{Linear}(1, 1/3, 25e6)$
IL and *IL+PPO* – Training	
# Processes to sample steps	40 (5 per GPU)
Common – *Training*	
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 $\texttt{Linear}(a, b, c)$ corresponds to linear interpolation between a and b within c training steps.

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

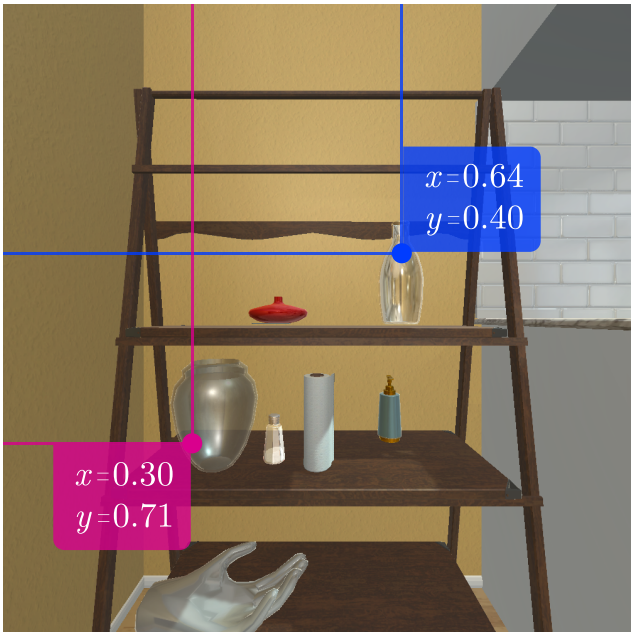

Figure 7: **Lower-level object targeting.** Instead of targeting objects based on their annotated type, the lower-level targeting action targets objects based on their location in the agent's current frame. For each (x, y) coordinate, with $0 \leq x, y \leq 1$, the x and y coordinates denote the relative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

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 $D : S \times S \to [0, 1]$, we define this energy function now. Let $s^i = (p^1, o^1, c^1, b^1), s^2 = (p^2, o^2, c^2, b^2) \in S$ 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
AlarmClock	✗	✓
AluminumFoil	✗	✓
Apple	✗	✓
ArmChair	✗	✗
BaseballBat	✗	✓
BasketBall	✗	✓
Bathtub	✗	✗
BathtubBasin	✗	✗
Bed	✗	✗
Blinds	✓	✗
Book	✓	✓
Boots	✗	✓
Bottle	✗	✓
Bowl	✗	✓
Box	✓	✓
Bread	✗	✓
ButterKnife	✗	✓
CD	✗	✓
Cabinet	✓	✗
Candle	✗	✓
CellPhone	✗	✓
Chair	✗	✗
Cloth	✗	✓
CoffeeMachine	✗	✗
CoffeeTable	✗	✗
CounterTop	✗	✗
CreditCard	✗	✓
Cup	✗	✓
Curtains	✗	✗
Desk	✗	✗
DeskLamp	✗	✗
Desktop	✗	✗
DiningTable	✗	✗
DishSponge	✗	✓
DogBed	✗	✗
Drawer	✓	✗
Dresser	✗	✗
Dumbbell	✗	✓
Egg	✗	✓
Faucet	✗	✗
Floor	✗	✗
FloorLamp	✗	✗
Footstool	✗	✓
Fork	✗	✓
Fridge	✓	✗
GarbageBag	✗	✗
GarbageCan	✗	✗
HandTowel	✗	✓
HandTowelHolder	✗	✗
HousePlant	✗	✗
Kettle	✓	✓
KeyChain	✗	✓
Knife	✗	✓
Ladle	✗	✓
Laptop	✓	✓
LaundryHamper	✓	✗
Lettuce	✗	✓
LightSwitch	✗	✗

Object Type [M-Z]	Openable	Pickupable
Microwave	✓	✗
Mirror	✗	✗
Mug	✗	✓
Newspaper	✗	✓
Ottoman	✗	✗
Painting	✗	✗
Pan	✗	✓
PaperTowelRoll	✗	✓
Pen	✗	✓
Pencil	✗	✓
PepperShaker	✗	✓
Pillow	✗	✓
Plate	✗	✓
Plunger	✗	✓
Poster	✗	✗
Pot	✗	✓
Potato	✗	✓
RemoteControl	✗	✓
RoomDecor	✗	✗
Safe	✓	✗
SaltShaker	✗	✓
ScrubBrush	✗	✓
Shelf	✗	✗
ShelvingUnit	✗	✗
ShowerCurtain	✓	✗
ShowerDoor	✓	✗
ShowerGlass	✗	✗
ShowerHead	✗	✗
SideTable	✗	✗
Sink	✗	✗
SinkBasin	✗	✗
SoapBar	✗	✓
SoapBottle	✗	✓
Sofa	✗	✗
Spatula	✗	✓
Spoon	✗	✓
SprayBottle	✗	✓
Statue	✗	✓
Stool	✗	✗
StoveBurner	✗	✗
StoveKnob	✗	✗
TVStand	✗	✗
TableTopDecor	✗	✓
TeddyBear	✗	✓
Television	✗	✗
TennisRacket	✗	✓
TissueBox	✗	✓
Toaster	✗	✗
Toilet	✓	✗
ToiletPaper	✗	✓
ToiletPaperHanger	✗	✗
Tomato	✗	✓
Towel	✗	✓
TowelHolder	✗	✗
VacuumCleaner	✗	✗
Vase	✗	✓
Watch	✗	✓
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.