(Supplementary Material)
A Simple Approach for Visual Room Rearrangement: 3D Mapping and Semantic Search

In this appendix we include the following supporting experiments and visualizations:

A. We begin this appendix by presenting the performance of our map disagreement detection module for each object category. We find that our method effectively detects map disagreements for both small and large objects, and is therefore robust to object size.

B. We then present a performance breakdown of our method for object size, distance to goal, and amount of clutter, and find that our method is less effective when objects are further from the goal or when nearby objects are closer together.

C. We report confidence intervals for our method's performance on the rearrangement challenge.

D. Finally, we outline the compute infrastructure needed to reproduce our experiments.

E. We list the hyperparameters used in our paper.

F. We categorize why our method can fail and provide a qualitative example.

The official code for our method will be released at publication.

A Object Type Versus Detection Accuracy

In this section, we visualize the relationship between the performance of our map disagreement detection module, detailed in Section 3, and the category of objects to be rearranged. For each of 1000 tasks in the validation set and test set of RoomR (Weihs et al., 2021), we record which object categories are detected as needing to be rearranged, and log the ground truth list of object categories that need to be rearranged. For each object, we calculate precision as the proportion of objects per category that were correctly identified as map disagreements out of all predicted map disagreements. Similarly, we calculate recall as the proportion of correctly identified as map disagreements out of all ground-truth map disagreements. Each bar in Figure 5 represents a 68% confidence interval of precision and recall over 1000 tasks per dataset split. The experiment shows that our method is robust to the size of objects that it rearranges because small objects such as the *SoapBar*, *CellPhone*, *CreditCard*, and *DishSponge* have comparable accuracy to large objects in Figure 5.

B Performance Analysis

This section extends Section 4.5 with an experiment to show potential failure modes. We consider three failure modes: (1) object size, (2) object distance to the goal, and (3) closest object in the same class. These indicators are visualized in Figure 6 against *%Fixed*. Our experiment suggests our method is robust to the size of objects, shown by the lack of a global trend in the left plot in Figure 6, and confirmed by Appendix A. Additionally, the experiment shows that objects further from the rearrangement goal are solved less frequently (middle plot), which is intuitive. Instances that have been shuffled to faraway locations in the scene may require longer exploration to find, and may be more difficult for our map disagreement detection module to match. A final conclusion we can draw from this experiment is that our method can fail when object instances are too close together. This is shown in the right plot in Figure 6 by the steep drop in performance when objects in the same category are < 1 meter apart. In this situation, our semantic mapping module can incorrectly detect two nearby objects as a single object, which prevents their successful rearrangement. For each of these potential failure modes, better perception and mapping approaches that more accurately describe object locations and appearance can improve fidelity of our method and reduce failure.

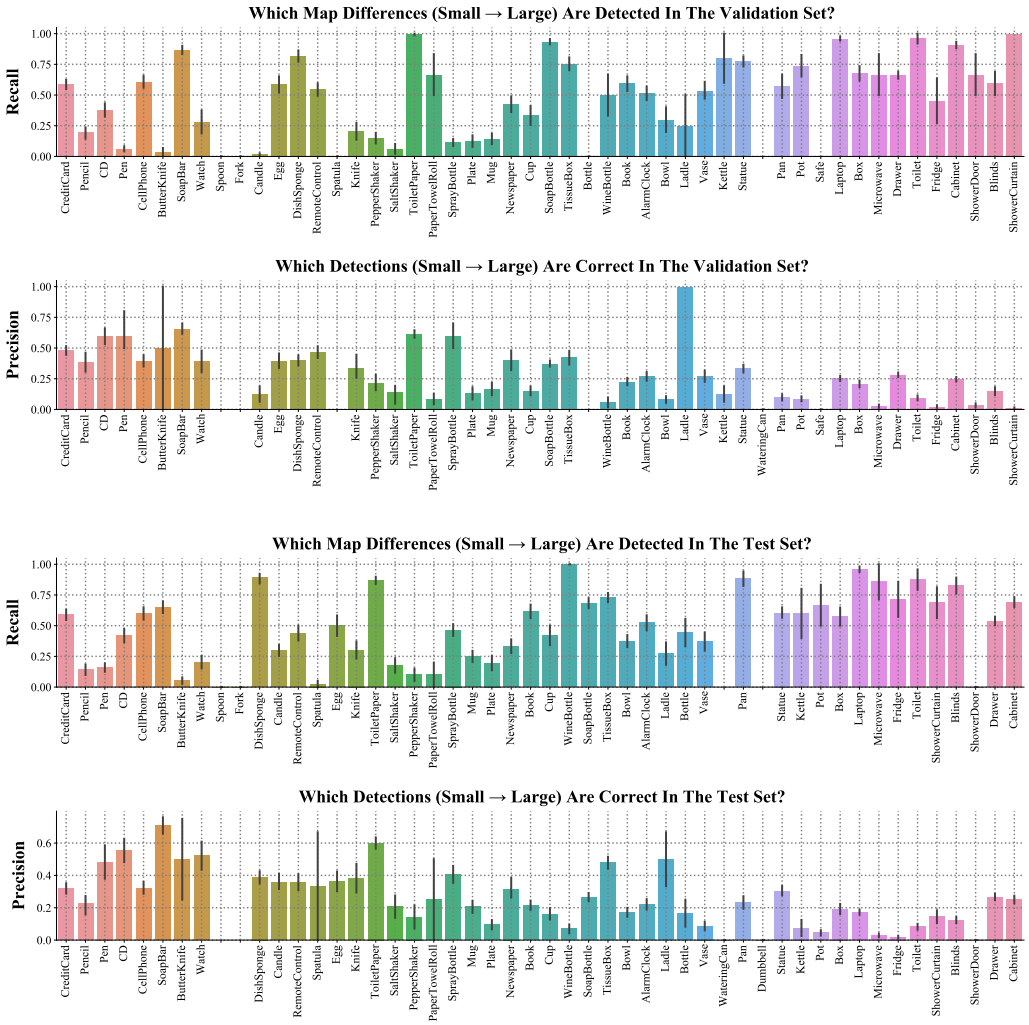

Figure 5: Performance breakdown on the validation and test sets for various types of objects. The height of bars corresponds to the sample mean of precision or recall for our map disagreement detection module. Error bars show a 68% confidence interval for each kind of object. The top two plots correspond to precision and recall on the validation set, while the bottom two plots correspond to precision and recall on the test set. Object categories are shown on the x-axis, and are ordered in ascending order of size. The experiment shows our method is robust to size, with small objects at the left end of the plots having comparable accuracy to large objects at the right end of the plots.

C Performance Confidence Intervals

We report 68% confidence intervals in Table 3 to supplement our evaluation in Section 4.1 and Section 4.2. We calculate intervals using 1000 tasks from the validation and test sets of the RoomR (Weihs et al., 2021) dataset, and report the mean followed by $\pm$ interval width. Note that the official rearrangement challenge leaderboard does not expose confidence intervals, nor the sample-wise performance needed to calculate them. Due to this, we are unable to compute confidence intervals of the baselines VRR (Weihs et al., 2021) and CSR (Gadre et al., 2022) at this time. These additional results show that our improvements over prior work significantly exceed the 68% confidence interval.

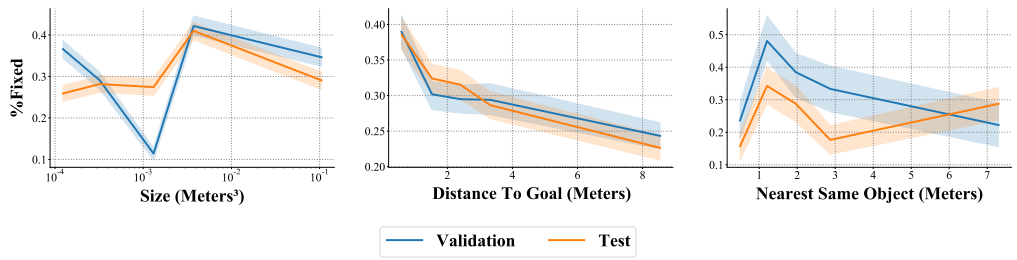

Figure 6: Performance of various ablations for different *Size (Meters*^{3}*)*, *Distance To Goal (Meters)*, and *Nearest Same Object (Meters)*. These indicators measure properties of objects that make rearrangement hard. Colored lines represent the average performance over 1000 tasks in each dataset split. Error bars represent a 68% confidence interval over those same 1000 sample points. The experiment shows our method can fail when objects of the same class are too close together (right plot), and when objects are too far from the goal location, typically >4.157 meters (center plot).

Table 3: Confidence intervals for our method on the AI2-THOR rearrangement challenge. Intervals are calculated from 1000 sample points from RoomR (Weihs et al., 2021) validation and test sets. We report pertformance starting with the sample mean, followed by $\pm$ a 68% confidence interval width. Our improvements over prior work significantly exceed the 68% confidence interval, which suggests that our improvements are significant and our method performs consistently well.

	Validation		**Test**	
Method	**%Fixed Strict**	**Success**	**%Fixed Strict**	**Success**
Ours w/o Semantic Search	15.77 ± 0.85	4.30 ± 0.63	15.11 ± 0.84	3.60 ± 0.58
Ours	17.47 ± 0.92	6.30 ± 0.76	16.56 ± 0.89	4.70 ± 0.67
Ours + GT Semantic Search	21.24 ± 0.99	7.60 ± 0.83	19.79 ± 0.96	6.10 ± 0.75
Ours + GT Segmentation	66.66 ± 1.21	45.60 ± 1.57	59.29 ± 1.26	37.55 ± 1.53
Ours + GT Both	68.46 ± 1.20	48.60 ± 1.57	59.50 ± 1.31	38.33 ± 1.57

D Required Compute

The goal of this section is to outline the amount of compute required to replicate our experiments. We will describe the amount of compute required for (1) training Mask R-CNN, (2) training a semantic search policy $\pi_\theta(\mathbf{x}|m_i)$, and (3) benchmarking the agent on the rearrangement challenge. For training Mask R-CNN, a dataset of 2 million images with instance segmentation labels were collected from the THOR simulator using the training split of the RoomR (Weihs et al., 2021) dataset. We then used Detectron2 (Wu et al., 2019b) with default hyperparameters to train Mask R-CNN with a ResNet50 (He et al., 2016) Feature Pyramid Network backbone (Lin et al., 2017). We trained our Mask R-CNN for five epochs using a DGX with eight Nvidia 32GB v100 GPUS for 48 hours. Our semantic search policy requires significantly less compute: completing 15 epochs on a dataset of 8000 semantic maps annotated with an expert search distribution in nine hours on a single Nvidia 12GB 3080ti GPU. Evaluating our method on the AI2-THOR rearrangement challenge requires 40 GPU-hours with a 2080ti GPU or equivalent. In practice, we parallelize evaluation across 32 GPUs, which results in an evaluation time of 1.25 hours for each of the validation and test sets.

E Hyperparameters

We provide a list of hyperparameters are their values in Table 4. These hyperparameters are held constant throughout the paper, except in ablations that study the sensitivity of our method to them, such as Section 4.3. Our ablations show our method is robust to these hyperparameters.

Table 4: Hyperparameters used by our approach for all rearrangement tasks.

Hyperparameter	**Value**
voxel size	0.05 meters
map height H	384
map width W	384
map depth D	96
classes C	54
detection confidence threshold	0.9
rearrangement distance threshold	0.05 meters
expert search distribution σ	0.75 meters
π_θ convolution hidden size	64
π_θ convolution kernel size	3×3
π_θ layers	5
π_θ activation function	ReLU
π_θ optimizer	Adam
π_θ learning rate	0.0003
π_θ batch size	8
π_θ epochs	15
π_θ dataset size	8000

F Reasons For Task Failures

This section explores the reasons why certain tasks in the validation and test sets are not solved by our method. We consider four reasons for task failures that cover all possible outcomes: (1) the agent correctly predicts which objects need to be moved where, but fails to rearrange at least one object, (2) the agent incorrectly predicts an object needs to be rearranged that doesn't, (3) the agent runs out of time, and (4) the agent misses at least one object that needs to be rearranged. We visualize the proportion of failed tasks for each category in Figure 7. We find that our method with ground truth perception and search (*Ours + GT Both*) tends to fail to rearrange objects after correctly identifying which objects need to be rearranged. In contrast, the largest reason for failure for our method (*Ours*) is the agent running out of time, followed by rearranging incorrect objects. This suggests the largest potential gains for our method arise from improving the speed and fidelity of map building, whereas, the optimality of the rearrangement policy becomes the bottleneck once a perfect map is available.

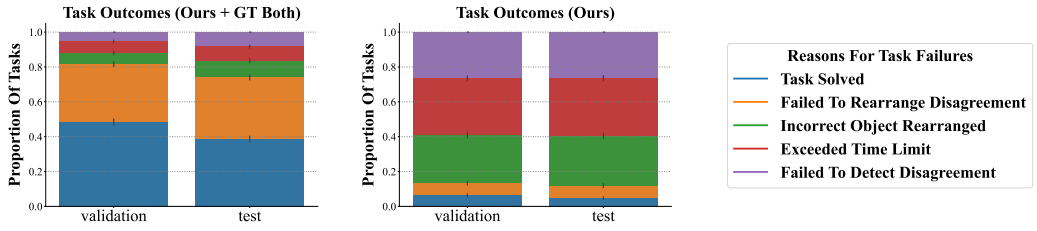

Figure 7: Categorization of the reasons why our method fails to solve tasks. The proportion of tasks that are solved (shown in blue) or fail due to one of four reasons (orange, green, red, purple) is shown for different ablations of our method. The height per bar corresponds to the proportion of tasks in the validation or test set in each category, and error bars indicate a 68% confidence interval. This experiment shows the largest reason for failure is a result of mapping errors. In the right plot, the agent fails most frequently by rearranging the wrong object, and by running out of time, which can result from imperfect semantic maps. In contrast, once perfect maps are available in the left plot, the largest source of errors are due to an imperfect planning-based rearrangement policy instead.

Locating *ToiletPaper* | Moving To Goal | Attempt Placing At Goal | Placed At Wrong Location

Figure 8: Qualitative example for why rearranging the correct object can fail. In this task, the agent correctly predicts the *ToiletPaper* needs to be rearranged, but fails to place the *ToiletPaper* in the correct location. The rightmost image shows the goal is located on the floor, but the agent mistakenly places the *ToiletPaper* on the bathtub instead, shown in the second image from the right.

G Image Features For Matching Object Instances

We conduct an experiment where we use image features for matching instances of objects between phases instead of their average color. We use ResNet50 (He et al., 2016) pretrained on ImageNet, and compute a mean image feature $h_i \in \mathcal{R}^{256}$ for each object. We process images with the ResNet50 backbone, extract a spatial feature map after the first residual block of ResNet50, and back-project the features to a voxel grid. In this fashion, we introduce 256 additional channels at each voxel to store the image features. We then compute h_i by averaging the voxel features corresponding to occupied voxels for object i in the map. Results in Table 5 show that matching object instances using image features improves the performance of our method by 6.97 %Fixed Strict on the test set.

Table 5: Matching object instances using image features. Results show that our method can be further improved by matching object instances between the walkthrough and unshuffle phases to maximize similarity of corresponding features from ResNet50 (He et al., 2016) pretrained on ImageNet.

	Validation		**Test**	
Method	**%Fixed Strict**	**Success**	**%Fixed Strict**	**Success**
Ours	17.47 ± 0.92	6.30 ± 0.76	16.62 ± 0.89	4.63 ± 0.67
Ours + Feature Matching	23.06 ± 1.04	7.81 ± 0.88	23.59 ± 1.03	6.79 ± 0.82

H Affect Of Semantic Search On Found Objects

To understand how our Semantic Search policy leads to an improvement in downstream performance, our hypothesis is that Semantic Search leads the agent to find more objects during episodes. We test this hypothesis by measuring the percent of objects found during episodes of the official 2022 Rearrangement Challenge. We consider an object found once the agent navigates within 1 meter of the object. We track the cumulative percent of objects found, and report in Figure 9 the mean and 68% confidence interval of this metric across 1000 test set tasks. The results confirm our hypothesis: in both phases, Semantic Search leads the agent to find more objects faster. In particular, at 250 episode timesteps during the Walkthrough phase, the agent has a 9.13 ± 1.23 higher percent than a uniform baseline. During the Unshuffle phase, the agent has a 6.23 ± 1.20 higher percent than our uniform baseline at 100 timesteps. This improvement becomes less significant as time increases during the Unshuffle phase, suggesting Semantic Search is most helpful with a small time budget.

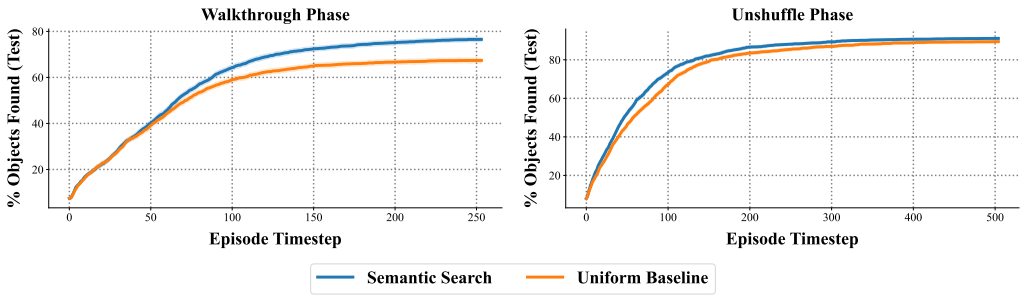

Figure 9: Impact of Semantic Search on the percent of shuffled objects found during either phase. In both cases, we observe an increase in the percent of shuffled objects found when using Semantic Search. The improvement is most significant during the walkthrough phase, where Semantic Search leads to an improvement of $+9.13$ percent of objects found at 250 episode timesteps. The improvement during the unshuffle phase is smaller, with $+6.23$ at 100 timesteps, and $+1.67$ at 500 timesteps.