Appendix to
Vision-based Navigation with Language-based Assistance via Imitation Learning with Indirect Intervention

1. Acknowledgements

We would like thank to Microsoft Research Redmond interns Xin Wang, Ziyu Yao, Shrimai Prabhumoye, Shi Feng, Amr Sharaf, Chen Zhao, and Evan Pu for useful discussions. We are grateful also to Michel Galley and Hal Daumé III for their advice and support. Special thanks goes to Vighnesh Shiv for his meticulous review of the paper. We also greatly appreciate the CVPR reviewers for their thorough and insightful comments.

2. Data Generation.

ASKNAV dataset. We partition all object instances into buckets where instances in the same bucket share the same environment, containing-room label, and object label. For each bucket, an end-goal is constructed as "Find [O] in [R]", where [O] is replaced with "a/an [object label]" (if singular) or "[object label]" (if plural), and [R] is replaced with "the [room label]" (if there is one room of the requested label) or "one of the `pluralize`([room label])"[1] (if there are multiple rooms of the requested label). We define the delegate viewpoint of an object as the closest viewpoint that is in the same room. To avoid annotation mistakes in the Matterport3D dataset, we ignore object instances that do not lie in the bounding boxes of their rooms. Goal viewpoints of an end-goal are delegate viewpoints of all object instances in the corresponding bucket. All viewpoints in the environment are candidates for the start viewpoint. We exclude candidates that are not reachable from any goal viewpoint and sample from the remaining candidates at most five start viewpoints per room. The initial heading angle is a random multiple of $\frac{\pi}{6}$ (less than 2π) and the initial elevation angle is always zero.

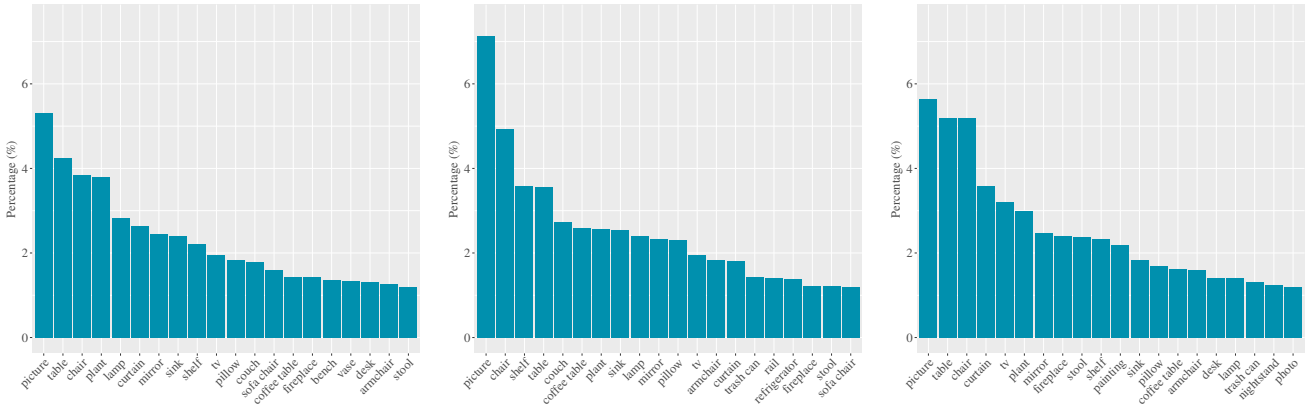
Each data point is defined as a tuple *(environment, start pose, goal viewpoints, end-goal)*. For each data point, we run the navigation teacher to obtain the sequences of actions that take the agent from the start viewpoint to the goal viewpoints. Note that executing those sequences of actions may not necessarily result in the agent facing the target objects in the end. We filter data points whose start viewpoints are adjacent to one of the goal viewpoints on the environment graph, or that require fewer than 5 or more than 25 actions to reach one of the goal viewpoints. We accumulate valid data points in all seen environments and sample a fraction to construct the seen sets. Algorithm 1 describes the sampling procedure. The remaining data points generated from the training environments form the training set. Similarly, the unseen sets are samples of data points in the unseen environments. We finally remove examples in the seen sets whose environments do not appear in the training set.

Algorithm 1 Data sampling procedure.

Input: a set of buckets $B = \{b_i\}$ where each bucket contains valid data points. N is the maximum number of elements to sample from each bucket ($N = 10$ for ASKNAV, $N = 20$ for NOROOM).
Output: a dataset D containing no less than 5000 data points.
Initialize $D = \emptyset$.
while $|D| < 5000$ and $B \neq \emptyset$ **do**
    Randomly shuffle elements of B.
    Mark all environments as 'not sampled'.
    for bucket b in B **do**
        $e \leftarrow$ environment of b.
        if e was 'not sampled' **then**
            s is a random sample of at most N elements of b.
            $D \leftarrow D \cup s$
            Remove b from B.
            Mark e as 'already sampled'.
        end if
    end for
end while

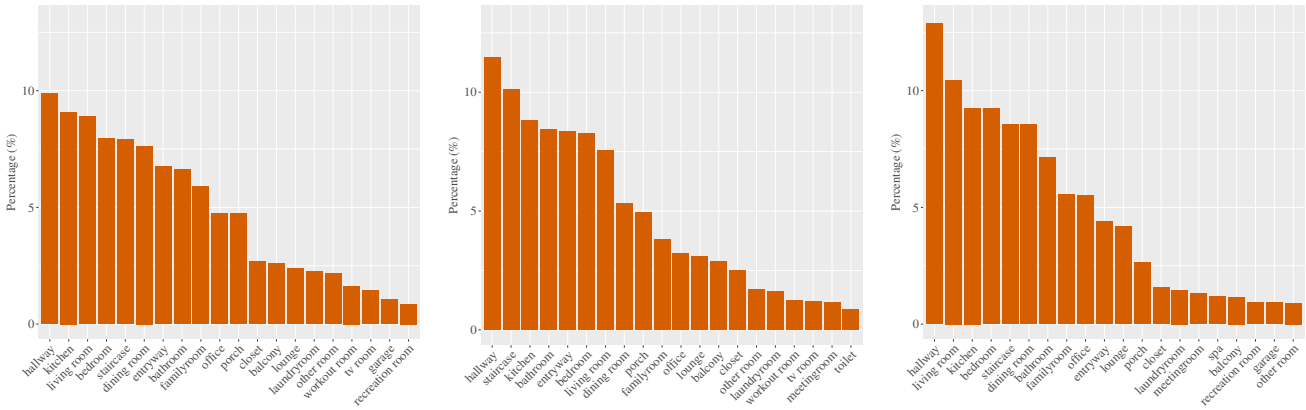
Figures 1, 3, 2, 4 offer more insights into the ASKNAV dataset. Most common target objects are associated with many instances in a house (e.g., picture, table, chair, curtain). Goal viewpoints mostly lie in rooms that contain many objects (e.g., bedroom, kitchen, living room, bathroom), whereas start viewpoints tend to be in the hallway because it is spacious and thus includes numerous viewpoints. About 85% of paths require at least ten actions to reach the goal viewpoints.

[1] We use https://github.com/jazzband/inflect to check for plurality and perform pluralization.



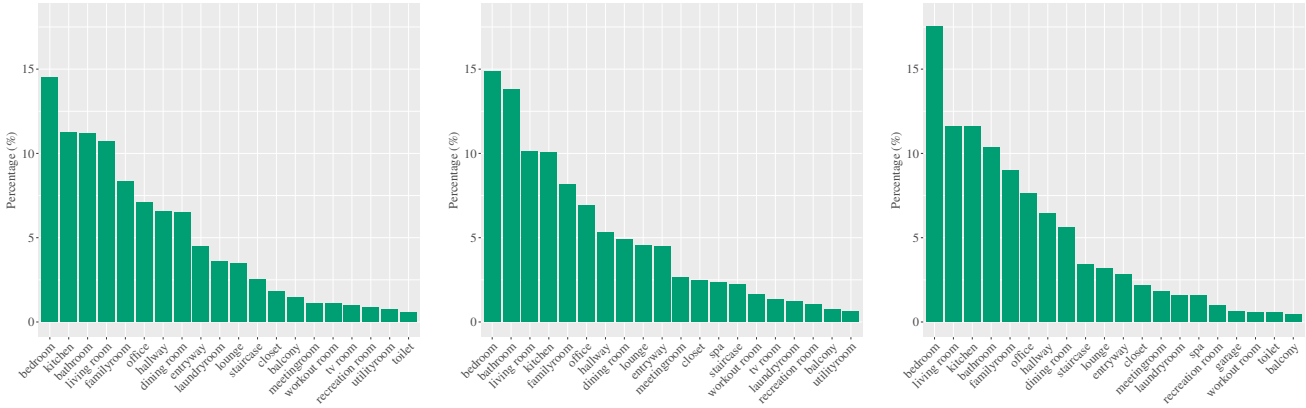
(a) Train (b) Test seen (c) Test unseen

Figure 1: Top 20 most common objects in the ASKNAV dataset.



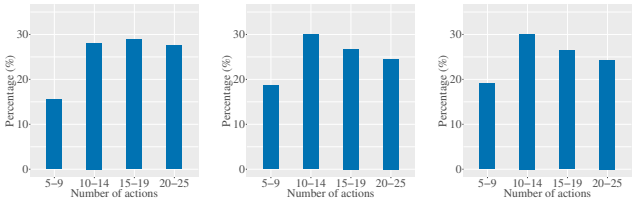
(a) Train (b) Test seen (c) Test unseen

Figure 2: Top 20 most common start rooms in the ASKNAV dataset.



(a) Train (b) Test seen (c) Test unseen

Figure 3: Top 20 most common goal rooms in the ASKNAV dataset.



(a) Train (b) Test seen (c) Test unseen

Figure 4: Distribution of path lengths in the ASKNAV dataset. Paths are computed by the shortest-path navigation teacher.

Split	Number of data points	Number of goals
Train	78,011	136,835
Dev seen	5,018	9,966
Dev unseen	5,003	9,318
Test seen	5,014	9,733
Test unseen	5,010	10,148

Table 1: NOROOM splits.

NOROOM dataset. The NOROOM data is generated using a similar procedure described above. However, since room types are not provided, each bucket is labeled with only the environment and the object label. End-goals have the form "Find [O]" instead of "Find [O] in [R]". To ensure that the number of goals of this dataset is comparable to that of the ASKNAV dataset, we sample at most 12 start viewpoints per object for each bucket (we sample five for ASKNAV). Table 1 summarizes the NOROOM dataset splits.

3. Time budget

We set the help-request budget B proportional to the time budget $\hat{T}$, which is the (approximate) number of actions required to reach the goal viewpoints. During training, for each data point, we set $\hat{T}$ to be the rounded average number of actions needed to move the agent along the shortest path from the start viewpoint to the goal viewpoints. During evaluation, because the shortest path needs to be unknown to the agent, we compute $\hat{T}$ based on the approximate number of actions to go optimally from the room type of the start viewpoint to the room type of the goal viewpoints[2]. This quantity is estimated using the training set.

Concretely, suppose we evaluate the agent on a data point d with starting viewpoint $\mathbf{x}_d^{\text{start}}$ and goal viewpoints $\{\mathbf{x}_{d,i}^{\text{end}}\}$. We define S as the multiset set of numbers of actions of training trajectories whose start and goal room types match those of d

$$S = \{\textsc{TrajLen}\left(\mathbf{x}_{d'}^{\text{start}}, \{\mathbf{x}_{d',i'}^{\text{end}}\}\right) : d' \in D, \quad (1)$$

$$r\left(\mathbf{x}_{d'}^{\text{start}}\right) = r\left(\mathbf{x}_{d}^{\text{start}}\right), r\left(\mathbf{x}_{d',i}^{\text{end}}\right) = r\left(\mathbf{x}_{d,i}^{\text{end}}\right)\} \quad (2)$$

where $\textsc{TrajLen}(.,.)$ returns the number of actions to move along the shortest path from a start viewpoint to a set of goal viewpoints, $r(.)$ returns the room type of a viewpoint, and D is the training set.

Next, we calculate the *95% upper confidence bound* of the mean number of actions

$$T = \begin{cases} \min(c_{\text{upper}}, L_{\text{max}}), \text{if } |S| > 0 \\ L_{\text{max}}, \qquad\qquad \text{if } |S| = 0 \end{cases} \quad (3)$$

$$c_{\text{upper}} = \text{mean}(S) + 1.95 \cdot \text{stdErr}(S)$$

mean(.) and stdErr(.) return the mean and standard error of a multiset, respectively, and L_{max} is a pre-defined constant. We then run the agent for $\hat{T} = \textsc{Round}(T)$ steps.

4. Hyperparameters

Table 2 summarizes hyperparameters used in our experiments. The navigation module uses unidirectional single-layer LSTMs as encoder and decoder. We initialize the encoder and the decoder by zero vectors. The help-requesting module is a feed-forward neural network with one hidden layer. We train the agent with Adam [1] for 10^5 iterations, using a learning rate of 10^{-4} without decaying and a batch size of 100. We regularize the agent with an L2-norm weight of 5×10^{-4} and a dropout ratio of 0.5. Training a LEARNED model took about 17 hours on a machine with a Titan Xp GPU and an Intel 4.00GHz CPU. The help-requesting ratio (τ) is 0.4 and the number of actions suggested by the subgoal advisor (k) is 4. The deviation threshold (δ), uncertainty threshold (ϵ), and non-moving threshold (μ) are 8, 1.0, and 9, respectively. The success radius (d) is fixed at 2 meters. Both the navigation and help-requesting modules are trained under the maximum log-likelihood objective, which maximizes the model-estimated probabilities of actions suggested by the teacher. We evaluate each agent with five different random seeds.

5. Qualitative Analysis

We analyze the behavior of an agent that is trained to learn a help-requesting policy (LEARNED) and is evaluated with a single random seed. This agent achieves a success rate of 52.0% on TEST SEEN and 34.5% on TEST UNSEEN. Overall, the success rate of the agent degrades as the trajectory gets longer (Figure 6). The agent tends to ask for help early (Figure 5), making more than half of its requests on the first 20% steps. As time advances, the number of requests decreases. As expected, the agent tends to request

[2]For the NOROOM dataset, we compute $\hat{T}$ based on the approximate number of actions to go from the start room type to the object type.

Hyperparameter	Value
Navigation module	
LSTM hidden size	512
Number of LSTM layers	1
Word embedding size	256
Navigation action embedding size	32
Help-requesting action embedding size	32
Budget embedding size	16
Image embedding size	2048
Coverage vector size	10
Help-requesting module	
Hidden size	512
Number of hidden layers	1
Activation function	RELU
Help-requesting teacher	
Deviation threshold (δ)	8
Uncertainty threshold (ϵ)	1.0
Non-moving threshold (μ)	9
Number of actions suggested by a subgoal (k)	4
Training	
Optimizer	Adam
Number of training iterations	10^5
Learning rate	10^{-4}
Learning rate decay	No
Batch size	100
Weight decay (L2-norm regularization)	5×10^{-4}
Dropout ratio	0.5
Help-requesting ratio (τ)	0.4
Evaluation	
Success radius (d)	2
Number of evaluating random seeds	5
Maximum time budget (L_{max})	25

Table 2: Hyperparameters.

help more early on TEST UNSEEN than on TEST SEEN. As shown in Figure 7, the most identifiable objects have distinct invariant features (e.g., sink, curtain), whereas the least identifiable objects greatly vary in shape and color (e.g., fireplace, stool, armchair). Mirrors are the easiest objects to detect in TEST SEEN, possibly because they are usually in small rooms (e.g., bathroom) and always reflect the camera used by the Matterport3D data collector. On TEST UNSEEN, they are more difficult to find because walking to the containing-rooms is more challenging. Finding objects in bathrooms is less challenging because bathrooms are usually small and have similar layouts and locations among houses, whereas searching for objects in offices is difficult because the corresponding environments are usually workspaces that contain many similar-looking rooms (Figure 8). Note that these rankings are subject to sampling biases; for example, they favor objects (or rooms) whose data points correspond to shorter paths.

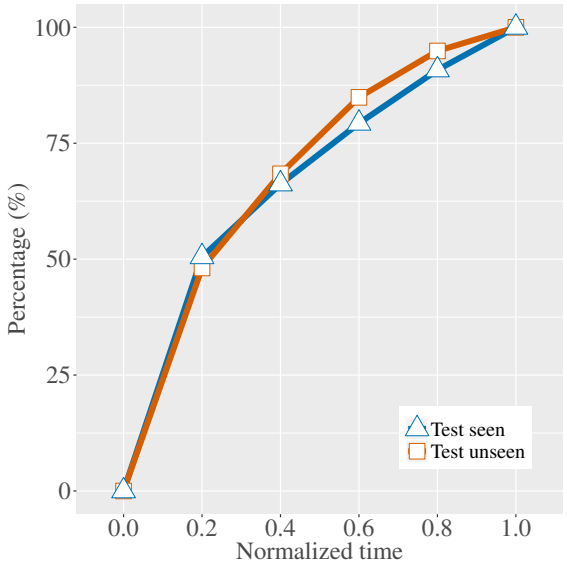


Figure 5: Fraction of help requests made over (normalized) time.

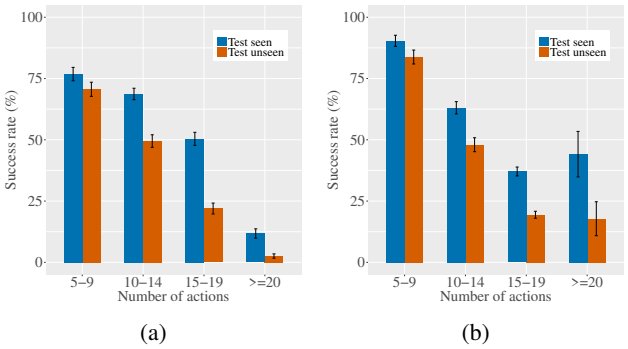
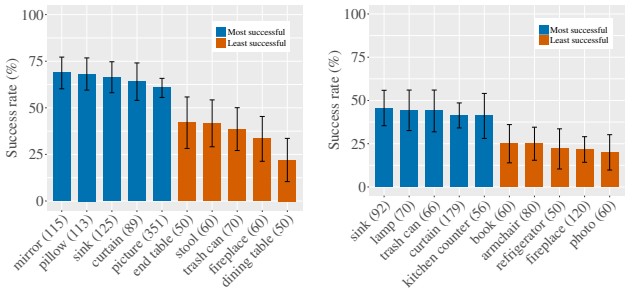


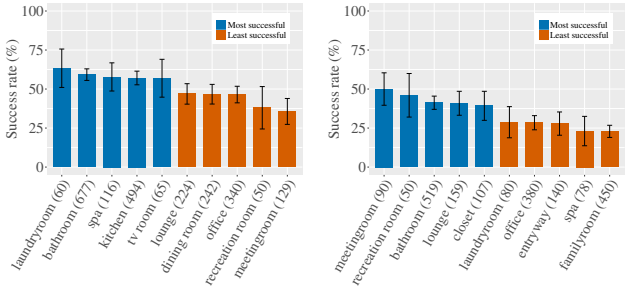
Figure 6: Success rate versus number of actions taken by (a) the navigation teacher and (b) the agent. Error bars are 95% confidence intervals.

Table 3 shows the effectiveness of different subsets of rules of the help-requesting teacher. Using only rules (b), (c), (d), which do not require learning because can be directly computed at test time without ground-truth information, is sufficient to obtain a success rate comparable to that of using all rules. Using rules (a) and (e), which require ground-truth information about the environment and the task, slightly improves the success rate over not requesting. Rules (a) and (e) are in fact very difficult to learn considering the small size of the Matterport3D dataset. Unfortunately, at the time this research was conducted, the Matterport3D simulator was one of the largest scale in the small pool of indoor simulators with real scenes. The effectiveness of rules (a) and (e) would be more visible in larger-scale environments like Gibson [2] but it offered limited



(a) (b)

Figure 7: Top five objects with highest and lowest average success rates in (a) TEST SEEN and (b) TEST UNSEEN. Numbers in parentheses are object frequencies. Only objects appearing more than 50 times are included. Error bars are 95% confidence intervals.



(a) (b)

Figure 8: Top five goal rooms with highest and lowest average success rates in (a) TEST SEEN and (b) TEST UNSEEN. Numbers in parentheses are room frequencies. Only rooms appearing more than 50 times are included. Error bars are 95% confidence intervals.

object annotation. Another reason that makes it challenging to learn rules (a) and (e) is the training-test condition mismatch. Near the end of training, the agent has memorized the training examples and rarely makes mistakes. The agent is thus biased toward not requesting help and generalizes poorly to unseen examples. We nevertheless include rules (a) and (e) to illustrate that the help-requesting policy can be taught rules that cannot be executed at test time by a teacher that has access to ground-truth information. In general, imitating a help-requesting teacher allows us to easily transfer domain knowledge from humans to the agent without restriction on the knowledge and on information required to imitate it.

Teacher	TEST SEEN	TEST UNSEEN
NONE	28.39 ± 0.00	6.36 ± 0.00
Rule (a), (e) ($\delta = 2$)	30.36 ± 0.13	8.49 ± 0.08
Rule (a), (e) ($\delta = 4$)	39.46 ± 0.05	8.78 ± 0.04
Rule (a), (e) ($\delta = 8$)	30.71 ± 0.05	5.64 ± 0.00
Rule (b), (c), (d)	51.89 ± 0.24	35.52 ± 0.29
All rules	52.09 ± 0.13	34.50 ± 0.23

Table 3: Ablation study on the effectiveness of rules of the help-requesting teacher. All numbers are success rates (%). See section 6.2 for specifications of the rules.

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015. 3

[2] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018. 4