

A TRAINING PROCEDURE

Training Algorithms. We pre-train the operation policy $\hat{\pi}$ with imitation learning (Ross et al., 2011), minimizing the cross entropy between its action distributions with those of a shortest-path oracle. We use advantage actor-critic (Mnih et al., 2016) to train the interaction policy ψ_θ . This method simultaneously estimates an actor policy $\psi_\theta : \bar{\mathcal{B}} \rightarrow \Delta(\bar{\mathcal{A}})$ and a critic function $V_\phi : \bar{\mathcal{B}} \rightarrow \mathbb{R}$. Given an execution $\bar{\tau} = (\bar{s}_1, \bar{a}_1, \bar{c}_1 \dots, \bar{s}_H)$, the gradients with respect to the actor and critic are computed as follows

$$\nabla_\theta \mathcal{L}_{\text{actor}} = \sum_{t=1}^H (V_\phi(\bar{b}_t^v) - C_t) \nabla_\theta \log \psi_\theta(\bar{a}_t | \bar{b}_t^a) \quad (7)$$

$$\nabla_\phi \mathcal{L}_{\text{critic}} = \sum_{t=1}^H (V_\phi(\bar{b}_t^v) - C_t) \nabla_\phi V_\phi(\bar{b}_t^v) \quad (8)$$

where $C_t = \sum_{j=t}^H c_j$, \bar{b}_t^a is a belief state summarizes the partial execution $\tau_{1:t}$ for the actor, and \bar{b}_t^v is a belief state for the critic.

Cost function. The cost function introduced in §4.4 is not effective for learning the interaction policy because the task error is given only at the end of an episode. We extend the reward-shaping method proposed by Ng et al. (1999) to goal-conditioned policies, augmenting the original cost function with a shaping function $\Phi(s, g)$ that measures the shortest-path distance from a location s to a goal g . The cost received by the agent at time step t is $\tilde{c}_t \triangleq \bar{c}_t + \Phi(s_{t+1}, g_{t+1}) - \Phi(s_t, g_t)$. We assume the agent transitions to and remains in a special terminal state $s_{\text{term}} \in \mathcal{S}$ after it terminates execution of the main goal. We set $\Phi(s_{\text{term}}, \text{None}) = 0$, where $g_t = \text{None}$ signals that the episode has ended. Hence, the cumulative cost of an execution under this new cost function is

$$\sum_{t=1}^H \tilde{c}_t = \sum_{t=1}^H \bar{c}_t + \Phi(s_{t+1}, g_{t+1}) - \Phi(s_t, g_t) = \sum_{t=1}^H \bar{c}_t - \Phi(s_1, g_1) \quad (9)$$

Since $\Phi(s_1, g_1)$ does not depend on the action taken in s_1 , minimizing the new cumulative cost does not change the optimal policy for the task (s_1, g_1) .

Model Architecture. We adapt the V&L BERT architecture (Hong et al., 2020) for modeling the operation policy $\hat{\pi}$. The model has two components: an encoder and a decoder; both are implemented as Transformer models (with self-attention). The encoder takes as input a description d_t^s or d_t^g and generates a sequence of hidden vectors. In every step, the decoder takes as input the previous hidden vector b_{t-1}^s , the sequence of vectors representing d_t^s , and the sequence of vectors representing d_t^g . It then performs self-attention on these vectors and computes the current hidden vector b_t^s and a probability distribution over navigation actions p_t .

The representation of each object is computed as follows. Let f^{name} , f^{horz} , f^{vert} , f^{dist} , and f^{type} are the name, horizontal angle, vertical angle, distance, and type of a room or object f (a type is either “room” or “object”). For simplicity, we discretize the feature values into 12 horizontal angles, 3 vertical angles, and 5 distances (by rounding down a real-valued distance to the nearest integer). For a room, f^{horz} , f^{vert} , f^{dist} are zeroes. We lookup the embedding of each feature from an embedding table and sum all the embeddings into a single vector that represents the corresponding room or object.

During pre-training, we randomly drop room or object features in d_t^s or d_t^g so that the navigation policy is familiar with making decision under incomplete information. Concretely, we define a *feature set* as all features of an object or the room name feature. For d_t^s , let M be the number objects in a description. We keep m feature sets where $m \sim \text{Uniform}(\min(5, M), M)$. For d_t^s , we have two cases. If g_1 is not within one node from s_1 , we uniformly randomly alternate between giving dense and sparse descriptions. Otherwise, when g_1 is adjacent or equals to s_1 , with a probability of $1/3$, we either give (a) a dense description (b) a (sparse) description that contains the goal room’s name and the target object’s features, or (c) a (sparse) description that describes the next ground-truth action.

The interaction policy ψ_θ is an LSTM-based recurrent neural network. The input of this model is navigation policy outputs, b_t^s and p_t , and the embedding of the previously taken action. The critic

Table 3: Dataset.

Split	Number of examples
Pre-training	82,104
Pre-training validation	3,000
Training	65,133
Validation UNSEENSTR	1,901
Validation UNSEENOBJ	1,912
Validation UNSEENENV	1,967
Test UNSEENSTR	1,653
Test UNSEENOBJ	1,913
Test UNSEENENV	1,777

Table 4: Hyperparameters.

Hyperparameter	Value
Environment	
Max. subgoal distance	3 nodes
Max. stack size	2
Max. object distance for d_t^s	5 meters
Max. object distance for d_t^g	3 meters
Max. number of objects (for dense d_t^s and d_t^g)	20
Cost of taking each CUR, GOAL, SUB, action	0.01
Operation policy $\hat{\pi}$	
Hidden size	256
Number of hidden layers	2
Attention dropout probability	0.1
Hidden dropout probability	0.1
Number of attention heads	8
Optimizer	Adam
Learning rate	10^{-4}
Batch size	32
Number of training iterations	10^5
Maximum number of time steps	15
Interaction policy ψ_θ	
Hidden size	512
Number of hidden layers	1
Entropy regularization weight	0.001
Optimizer	Adam
Learning rate	10^{-5}
Batch size	32
Number of training iterations	5×10^4
Maximum number of time steps	30

model also has a similar architecture but outputs a real number (the V value) rather than an action distribution. When training the interaction policy, we always fix the parameters of the navigation policy. We find it necessary to pre-train the critic policy before training it jointly with the actor policy.

Data. See Table 3 for a summary of the data splits. From a total of 72 environments provided the Matterport3D dataset, we use 36 environments for pre-training, 18 as unseen during training, 11 for validation UNSEENENV, and 7 for test UNSEENENV. We use a vocabulary of size 1738, which include object and room names, and special tokens for distances and directions. Each navigation path in our dataset has from 5 to 10 nodes.

Hyperparameters. See Table 4.