Appendix to
Help, Anna! Visual Navigation with Natural Multimodal Assistance via Retrospective Curiosity-Encouraging Imitation Learning

Acknowledgement

We would like to thank Debadeepta Dey for helpful discussions. We thank Jesse Thomason for the thorough review and suggestions on related work and future directions. Many thanks to the reviewers for their meticulous and insightful comments. Special thanks to Reviewer 2 for addressing our author response in detail, and to Reviewer 3 for the helpful suggestions and the careful typing-error correction.

A Proof of Lemma 1

Lemma 1. *To guide the agent between any two locations using $O(\log N)$ instructions, we need to collect instructions for $\Theta(N \log N)$ location pairs.*

Proof. Consider a spanning tree of the environment graph that is rooted at v_r. For each node v, let $d(v)$ be the distance from v to v_r. For $i = 0 \cdots \lfloor \log_2 d(v) \rfloor - 1$, collect an instruction for the path from v to its 2^i-th ancestor, and an instruction for the reverse path. In total, no more than $2 \cdot N \log_2 N = \Theta(N \log N)$ instructions are collected.

A language-assisted path is a path that is associated with a natural language instruction. It is easy to show that, under this construction, $O(\log N)$ language-assisted paths are sufficient to traverse between v and any ancestor of v as follows. For any two arbitrary nodes u and v, let w be their least common ancestor. To traverse from u to v, we first go from u to w, then from w to v. The total number of language-assisted path is still $O(\log N)$. □

B Proof of Lemma 2

Lemma 2. *At any time step, the retrospective help-request teacher suggests the action that results in the agent getting closer to the target location in the future under its current navigation policy (if such an action exists).*

The retrospective mode of interaction allows the teacher to observe the agent's full trajectory when computing the reference actions. Note that this trajectory is generated by the agent's own navigation policy ($\hat{\pi}_{\text{nav}}$) because, during training, we use the agent's own policies to act. During a time step, after observing the future trajectory, the teacher determines whether the `lost` condition is satisfied or not, i.e. whether the agent will get closer to the target location or not.

1. If condition is *not* met, the teacher suggests the `do_nothing` action: it is observed that the agent will get closer the target location without requesting help by following the current navigation policy.
2. If condition is met, the teacher suggests the `request_help` action. First of all, in this case, it is observed that the `do_nothing` action will lead to no progress toward the target location. If the `request_help` action will also lead to no progress, then it does not matter which action the teacher suggests. In contrast, if the `request_help` action will help the agent get closer to the target location, the teacher has suggested the desired action.

C Features

At time step t, the model makes use of the following features:

- $\diamond$ I_t^{cur}: set of visual feature vectors representing the current panoramic view;
- $\diamond$ I_t^{tgt}: set of visual feature vectors representing the target location's panoramic view;
- $\diamond$ a_{t-1}: embedding of the last action;
- $\diamond$ δ_t: a vector representing how similar the target view is to the current view;

- η_t^{loc}: local-time embedding encoding the number of time steps since the last time the inter-task module was reset;
- η_t^{glob}: global-time embedding encoding the number time steps since the beginning of the episode;
- P_t^{nav}: the navigation action distribution output by $\hat{\pi}_{\text{nav}}$. Each action corresponding to a view angle is mapped to a static index to ensure that the order of the actions is independent of the view angle. This feature is only used by the help-request network.

Visual features. To embed the first-person view images, we use the visual feature vectors provided Anderson et al. (2018), which are extracted from a ResNet-152 (He et al., 2016) pretrained on ImageNet (Russakovsky et al., 2015). Following the Speaker-Follower model (Fried et al., 2018), at time step t, we provide the agent with a feature set I_t^{cur} representing the current panoramic view. The set consists of visual feature vectors that represent all 36 possible first-person view angles (12 headings $\times$ 3 elevations). Similarly, the panoramic view at the target location is given by a feature set I_t^{tgt}. Each next location is associated with a view angle whose center is closest to it (in angular distance). The embedding of a navigation action $(v, \Delta\psi, \Delta\omega)$ is constructed by concatenating the feature vector of the corresponding view and an orientation feature vector $[\sin\Delta\psi; \cos\Delta\psi; \sin\Delta\omega; \sin\Delta\omega]$ where $\Delta\psi$ and $\Delta\omega$ are the camera angle change needed to shift from the current view to the view associated with v. The `stop` action is mapped to a zero vector. The action embedding set E_t^{nav} contains embeddings of all navigation actions at time step t.

Target similarity features. The vector δ represents the similarity between I^{cur} and I^{tgt}. To compute this vector, we first construct a matrix C, where $C_{i,j}$ is the cosine similarity between I_i^{cur} and I_j^{tgt}. δ_t is then obtained by taking the maximum values of the rows of C. The intuition is, for each current view angle, we find the most similar target view angle. If the current view perfectly matches with the target view, δ_t is a vector of ones.

Time features. The local-time embedding is computed by a residual neural network (He et al., 2016) that learns a time-incrementing operator

$$\eta_t^{\text{loc}} = \textsc{IncTime}\left(\eta_{t-1}^{\text{loc}}\right) \quad (1)$$

where $\textsc{IncTime}(x) = \textsc{LayerNorm}(x + \textsc{ReLU}(\textsc{Linear}(x)))$. The global-time embedding is computed similarly but also incorporates the current local-time

$$\eta_t^{\text{glob}} = \textsc{IncTime}\left(\left[\eta_t^{\text{loc}}; \eta_{t-1}^{\text{glob}}\right]\right) \quad (2)$$

The linear layers of the local and global time modules do not share parameters. Our learned time features generalizes to previously unseen numbers of steps. They allow us to evaluate the agent on longer episodes than during training, significantly reducing training cost. We use the sinusoid encoding (Vaswani et al., 2017) for the text-encoding module, and the ResNet-based encoding for the decoder modules. In our preliminary experiments, we also experimented using the sinusoid encoding in all modules but doing so significantly degraded success rate.

D Model

Modules. The building blocks of our architecture are the Transformer modules (Vaswani et al., 2017)

- $\textsc{TransEncoder}(l)$ is a Transformer-style encoder, which generates a set of feature vectors representing an input sequence l;
- $\textsc{MultiAttend}(q, K, V)$ is a multi-headed attention layer that takes as input a query vector q, a set of key vectors V, and a set of value vectors V. The output is added a residual connection (He et al., 2016) and is layer-normalized (Lei Ba et al., 2016).
- $\text{FFN}(x)$ is a Transformer-style feed-forward layer, which consists of a regular feed-forward layer with a hidden layer and the ReLU activation function, followed by a residual connection and layer normalization. The hidden size of the feed-forward layer is four times larger than the input/output size.

In addition, we devise the following new attention modules

- $\textsc{SelfAttend}(q, K, V)$ implements a multi-headed attention layer internally. It calculates an output $h = \textsc{MultiAttend}(q, K, V)$. After that, the input q and output h are appended to K and V, respectively. This module is different from the Transformer's self-attention in that the keys and values are distinct.
- $\textsc{SimAttend}(q, K, V)$ computes a weighted value $h = \sum_i \tilde{a}_i V_i$ where each weight $\tilde{a}_i$ is

Hyperparameter	Value
Common	
Hidden size	256
Navigation action embedding size	256
Help-requesting action embedding size	128
Word embedding size	256
Number of self-attention heads	8
Number of instruction-attention heads	8
ResNet-extracted visual feature size	2048
Help-request teacher	
Uncertainty threshold (γ)	0.25
Training	
Optimizer	Adam
Number of training iterations	3×10^4
Learning rate	10^{-4}
Batch size	32
Dropout ratio	0.3
Training time steps	20
Maximum instruction length	50
Curiosity-encouraging weight (α)	$1.0^{(*)}$
Evaluation	
Success radius ($\epsilon_{\text{success}}$)	2 meters
Attention radius (ϵ_{attn})	2 meters
Evaluation time steps	50

Table 1: Hyperparameters. (*) Training collapses when using $\alpha = 1$ to train the agent with the help-request policy baselines (NOASK, RANDOMASK, ASKEVERY5). Instead, we use $\alpha = 0.5$ in those experiments.

defined as

$$\tilde{a}_i = \mathbb{I}\{a_i > 0.9\} \frac{a_i}{\sum_j a_j} \quad (3)$$

$$a_i = \textsc{CosineSimilarity}(q, K_i) \quad (4)$$

where $\textsc{CosineSimilarity}(.,.)$ returns the cosine similarity between two vectors, and $\mathbb{I}\{.\}$ is an indicator function. Intuitively, this module finds keys that are nearly identical to the query and returns the weighted average of values corresponding to those keys. We use this module to obtain a representation of related past, which is crucial to enforcing curiosity-encouraging training.

We now describe the navigation network in detail. For notation brevity, we omit the $^{\text{nav}}$ superscripts in all variables.

Text-encoding component. The agent maintains a text memory M^{text}, which stores the hidden representation of the current language instruction l_t. At the beginning of time, the agent encodes the task request to generate an initial text memory. During time step t, if the current task is altered (due to the agent requesting help or departing a route), the agent encodes the new language instruction to generate a new text memory

$$M^{\text{text}} = \textsc{TransEncoder}(l_t) \quad \text{if } l_t \neq l_{t-1} \text{ or } t = 0 \quad (5)$$

Inter-task component. The inter-task module computes a vector h_t^{inter} representing the state of the current task's execution. This state and the local time embedding are reset to zero vectors every time the agent switches task[1]. Otherwise, a new state is computed as follows

$$h_t^{\text{inter}} = \textsc{SelfAttend}(q_t^{\text{inter}}, M_{\text{in}}^{\text{inter}}, M_{\text{out}}^{\text{inter}}) \quad (6)$$

$$q_t^{\text{inter}} = W_{\text{inter}}[c_t^{\text{inter}}; a_{t-1}; \delta_t] + \eta_t^{\text{loc}} \quad (7)$$

$$c_t^{\text{inter}} = \textsc{DotAttend}(h_{t-1}^{\text{inter}}, I_t^{\text{cur}}) \quad (8)$$

where $M_{\text{in}}^{\text{inter}} = \{q_0^{\text{inter}}, \cdots, q_{t-1}^{\text{inter}}\}$ and $M_{\text{in}}^{\text{inter}} = \{h_0^{\text{inter}}, \cdots, h_{t-1}^{\text{inter}}\}$ are the input and output inter-task memories, and $\textsc{DotAttend}(q, M)$ is the dot-product attention (Luong et al., 2015).

Next, h_t^{inter} is used to select which part of the language instruction should be interpreted in this step

$$c_t^{\text{text}} = \text{FFN}\left(\tilde{c}^{\text{text}}\right) \quad (9)$$

$$\tilde{c}^{\text{text}} = \textsc{MultiAttend}(h_t^{\text{inter}}, M_t^{\text{text}}, M_t^{\text{text}}) \quad (10)$$

Finally, c_t^{text} is used to select which areas of the current view and target view the agent should focus on

$$c_t^{\text{cur}} = \textsc{DotAttend}(c_t^{\text{text}}, I_t^{\text{cur}}) \quad (11)$$

$$c_t^{\text{tgt}} = \textsc{DotAttend}(c_t^{\text{text}}, I_t^{\text{tgt}}) \quad (12)$$

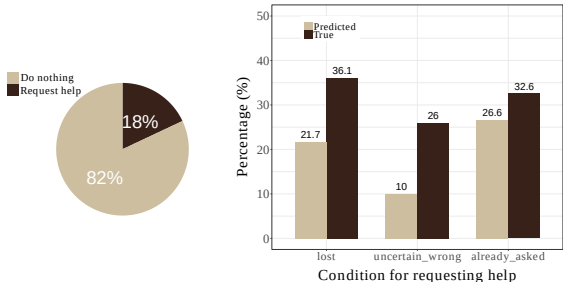
Intra-task component. The inter-task module computes a vector h_t^{intra} representing the state of the entire episode. To compute this state, we first calculate $\bar{h}_t^{\text{intra}}$, a tentative current state, and $\tilde{h}_t^{\text{intra}}$, a weighted combination of the past states at nearly identical situations

$$\bar{h}_t^{\text{intra}} = \text{FFN}\left(\textsc{SelfAttend}(q_t^{\text{intra}}, M_{\text{in}}^{\text{intra}}, M_{\text{out}}^{\text{intra}})\right) \quad (13)$$

$$\tilde{h}_t^{\text{intra}} = \textsc{SimAttend}(q_t^{\text{intra}}, M_{\text{in}}^{\text{intra}}, M_{\text{out}}^{\text{intra}}) \quad (14)$$

$$q_t^{\text{intra}} = W_{\text{intra}}\left[c_t^{\text{text}}; c_t^{\text{cur}}; c_t^{\text{tgt}}; \delta_t\right] + \eta_t^{\text{glob}} \quad (15)$$

[1] In practice, every time we reset the state and the local time embedding in an episode, we also reset all states and local time embeddings in all episodes in the same batch.



(a) (b)

Figure 1: Help-request behavior on TEST UNSEENALL: (a) fraction of time steps where the agent requests help and (b) predicted and true condition distributions. The `already_asked` condition is the negation of the `never_asked` condition.

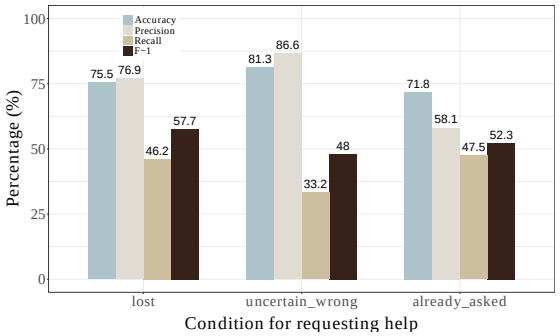


Figure 2: Accuracy, precision, recall, and F-1 scores in predicting the help-request conditions on TEST UNSEENALL. The `already_asked` condition is the negation of the `never_asked` condition.

where $M_{\text{in}}^{\text{intra}} = \{q_0^{\text{intra}}, \cdots, q_{t-1}^{\text{intra}}\}$ and $M_{\text{in}}^{\text{intra}} = \{h_0^{\text{intra}}, \cdots, h_{t-1}^{\text{intra}}\}$ are the input and output intra-task memories. The final state is determined by subtracting a scaled version of $\tilde{h}_t^{\text{intra}}$ from $\bar{h}_t^{\text{intra}}$

$$h_t^{\text{intra}} = \bar{h}_t^{\text{intra}} - \beta \cdot \tilde{h}_t^{\text{intra}} \quad (16)$$

$$\beta = \sigma(W_{\text{gate}} \cdot [\bar{h}_t^{\text{intra}}; \tilde{h}_t^{\text{intra}}]) \quad (17)$$

Finally, the navigation action distribution is computed as follows

$$P_t^{\text{nav}} = \text{SOFTMAX}(W^{\text{out}}\, h_t^{\text{intra}}\, W^{\text{act}}\, E_t^{\text{nav}})) \quad (18)$$

where E_t^{nav} is a matrix containing embeddings of the navigation actions. The computations in the help-request network is almost identical except that (a) the navigation action distribution P_t^{nav} is fed as an extra input to the intra-task components (15), and (b) the help-request and reason distributions are calculated as follows

$$P_t^{\text{ask}} = \text{SOFTMAX}(E_t^{\text{ask}} E_t^{\text{reason}} h_t^{\text{ask,intra}}) \quad (19)$$

$$P_t^{\text{reason}} = \text{SOFTMAX}(E_t^{\text{reason}} h_t^{\text{ask,intra}}) \quad (20)$$

where E_t^{ask} contains embeddings of the help-request actions and E_t^{reason} contains embeddings of the reason labels.

E Hyperparameters

Table 1 summarizes all training and evaluation hyperparameters. Training our agent takes approximately 9 hours on a single Titan Xp GPU.

F Analysis

Ablation Study. Table 2 shows an ablation study on the techniques we propose in this paper. We observe the performance on VAL SEENENV of the model trained without the curiosity-encouraging loss ($\alpha = 0$) is slightly higher than that of our final model, indicating that training with the curiosity-encouraging loss slightly hurts memorization. This is expected because the model has relatively small size but has to devote part of its parameters to learn the curiosity-encouraging behavior. On the other hand, optimizing with the curiosity-encouraging loss helps our agent generalize better to unseen environments. Predicting help-request conditions produces contrasting effects on the agent in seen and unseen environments, boosting performance in VAL SEENENV while slightly degrading performance in VAL UNSEENALL. We investigate the help-request patterns and find that, in both types of environments, the agent makes significantly more requests when not learning to predict the conditions. Specifically, the no-condition-prediction agent meets the `uncertain_wrong` condition considerably more often (+5.2% on VAL SEENENV and +6.4% on VAL UNSEENALL), and also requests help at a previously visited location while executing the same task more frequently (+7.22% on VAL SEENENV and +8.59% on VAL UNSEENALL). This phenomenon highlights a challenge in training the navigation and training the help-request policies jointly: as the navigation policy gets better, there are less positive examples (i.e., situations where the agent needs help) for the help-request policy to learn from. In this specific case, learning a navigation policy that is less accurate in seen environments is beneficial to the agent when it encounters unseen environments because such a navigation policy creates more positive examples for training

Agent	VAL SEENENV				VAL UNSEENALL			
	SR ↑ (%)	SPL ↑ (%)	Nav. ↓ Err. (m)	Requests/ task ↓	SR ↑ (%)	SPL ↑ (%)	Nav. ↓ Err. (m)	Requests/ task ↓
Final agent	87.24	63.02	**1.21**	2.9	45.64	22.68	7.72	**5.9**
No inter-task reset	83.62	60.44	1.44	3.2	40.60	19.89	8.26	7.3
No condition prediction	72.33	47.10	2.64	4.7	**47.88**	**24.68**	**6.61**	7.9
No cosine-similarity attention ($\beta = \mathbf{0}$)	83.08	59.52	1.68	3.0	43.69	22.44	7.94	6.9
No curiosity-encouraging loss ($\alpha = 0$)	**87.36**	**70.18**	1.25	**2.0**	39.23	20.78	8.88	7.5

Table 2: Ablation study on our proposed techniques. Models are evaluated on the validation splits. Best results in each column are in bold.

the help-request. Devising learning strategies that learns both efficient navigation and help-request policies is an exciting future direction.

Help-request Behavior on TEST UNSEENALL. Our final agent requests about 18% of the total number of time steps (Figure 1a). Overall, it learns a conservative help-request policy (Figure 1b). Figure 2 shows how accurate our agent predicts the help-request conditions. The agent achieves high precision scores in predicting the `lost` and `uncertain_wrong` conditions (76.9% and 86.6%), but achieves lower recalls in all conditions (less than 50%). Surprisingly, it predicts the `already_asked` condition less accurate than the other two, even though this condition is intuitively fairly simple to realize.

References

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. 2018. Speaker-follower models for vision-and-language navigation. In *Proceedings of Advances in Neural Information Processing Systems*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of Emperical Methods in Natural Language Processing*.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.