

Supplementary Material: Topological Planning with Transformers for Vision-and-Language Navigation

1. Topological Maps

1.1. Exploration Trajectories

As a first step prior to navigation, the agent builds an understanding of the environment in the form of a topological map. To do this, it uses a set of exploration trajectories which can be derived using a variety of methods such as node classification [4], frontier exploration [16], or other approaches. In our work, we generate the exploration trajectories by sampling a set of waypoints in the traversable areas of the environments using the ground truth metric maps and then having the agent navigate to each of them. This allows us to create a fixed set of exploration trajectories to make the experiments and evaluation across baselines consistent. During the exploration process the agent keeps track of its observation at each time step as well as the odometry sensor reading of its position and orientation such that the agent is aware of the relative position between any two observations. The result is a dense trajectory throughout the environment as visualized in Fig. 1.

1.2. Map Construction

1.2.1 Reachability Estimator

Examples of our agent-generated topological maps are shown in Fig. 2. We use a reachability estimator [10] to sparsify the exploration trajectories into a topological map. A reachability estimator is a function RE that predicts the probability of any given controller to start from a certain position and reach a certain target. In our work, we use a simple oracle reachability estimator RE that exploits the traversability of the environment. Here, we define reachability between two positions p_1 and p_2 as:

$$RE(p_1, p_2) = \delta(p_1, p_2) * \max\left(\frac{d_{sp} - \|p_1 - p_2\|}{d_{sp}}, 0\right) \quad (1)$$

where $\delta(p_1, p_2)$ represents line-of-sight between p_1 and p_2 according to the traversability maps (as shown in white and gray in Fig. 1) and d_{sp} is a parameter representing the sparsity of the resulting topological maps. In other words, reachability is nonzero if an unobstructed line can be drawn between p_1 and p_2 and if they are within d_{sp} from each other.

1.2.2 Sparsifying a dense trajectory

Given a dense exploration trajectory, for any starting position p_i , we may confidently discard any position $p_{i+1} \dots p_{j-1}$ as long as $RE(p_i, p_j)$ is sufficiently high [10]. In other words, we greedily choose the next node by

$$\begin{aligned} & \max j \\ & \text{s.t. } RE(p_i, p_j) > p_{sparse}, \forall k, i < k \leq j \end{aligned} \quad (2)$$

where p_{sparse} is the threshold that guarantees a minimum reachability of any edge in the resultant graph.

Each exploration trajectory produces one sparsified topological map. Since we perform multiple exploration runs per environment, we merge multiple topological maps into one by adding edges between any two node i and j similarly if $RE(p_i, p_j) > p_{sparse}$.

Finally, we further sparsify the resultant singular graph by merging nodes i and j if $RE(p_i, p_j) > p_{merge}$, where p_{merge} is a higher threshold above which two nodes are merged into a single one. All neighbors of i and j become the neighbors of the new node, except the ones with reachability dropping below p_{sparse} after the merge.

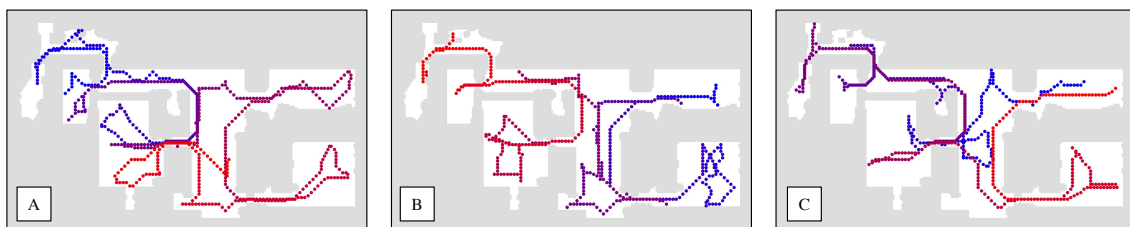
In our setup, we use $d_{sp} = 4m$, $p_{sparse} = 0$, $p_{merge} = 0.5$, such that edges are established between positions reachable in a straight line within $4m$ of each other, and nodes within $2m$ of each other are merged.

1.2.3 Comparison to Pre-Defined R2R Graphs

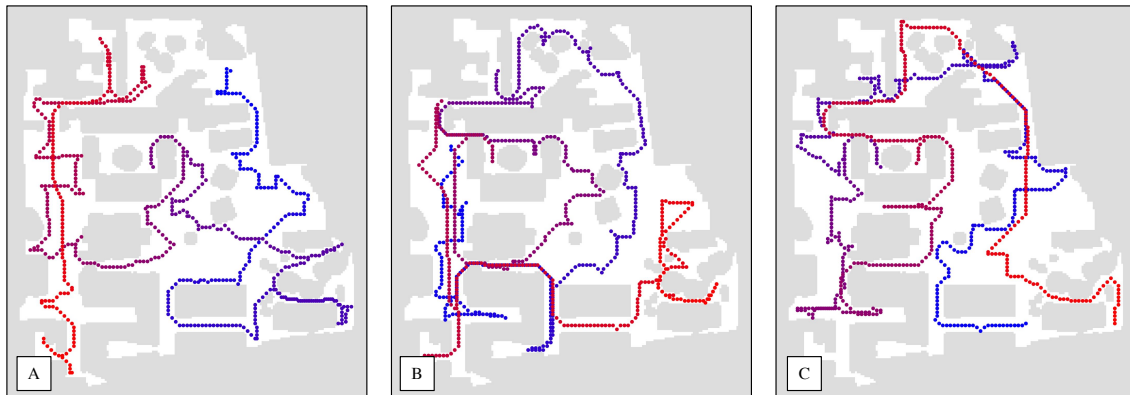
We illustrate differences between our agent-generated (AG) maps and the predefined Room2Room (R2R) graphs [1] in Fig. 3. The AG maps are sparser and may have nodes which are very close to the walls. For instance, as shown in the top row of Fig. 3, in the R2R graph the hallway along the center of the map has nicely positioned nodes. However, the nodes in the AG maps for the same space are positioned along the walls, resulting in slightly more redundant nodes.

We also see large node clusters in open spaces in the R2R graphs. In these locations, there are several nearby nodes in the same area as well as edges which connect the nodes with each other. This can be seen in every R2R graph in Fig. 3.

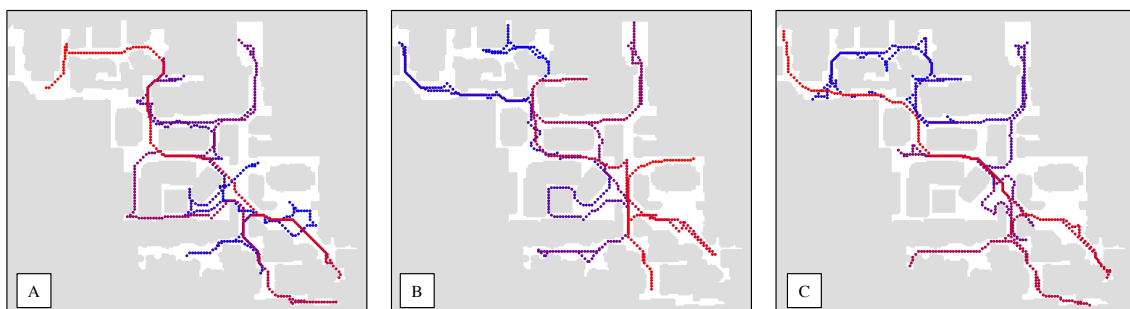
Environment 1



Environment 2



Environment 3



Environment 4

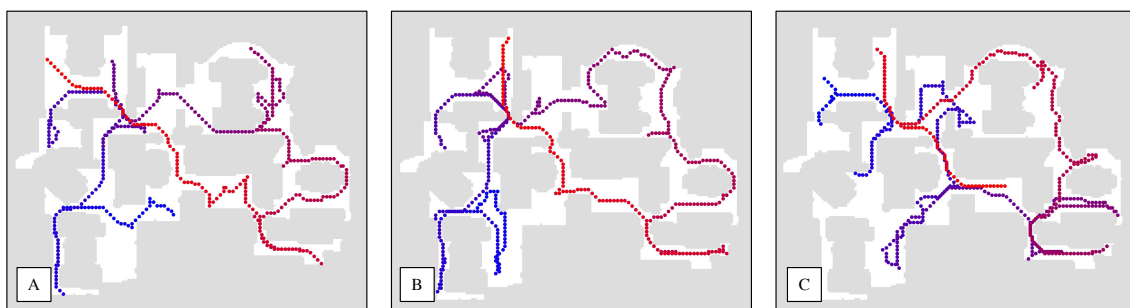


Figure 1: **Exploration trajectories.** Each row shows different exploration trajectories, going from blue to red, for a single environment. White represents traversable regions, while gray represents untraversable areas.

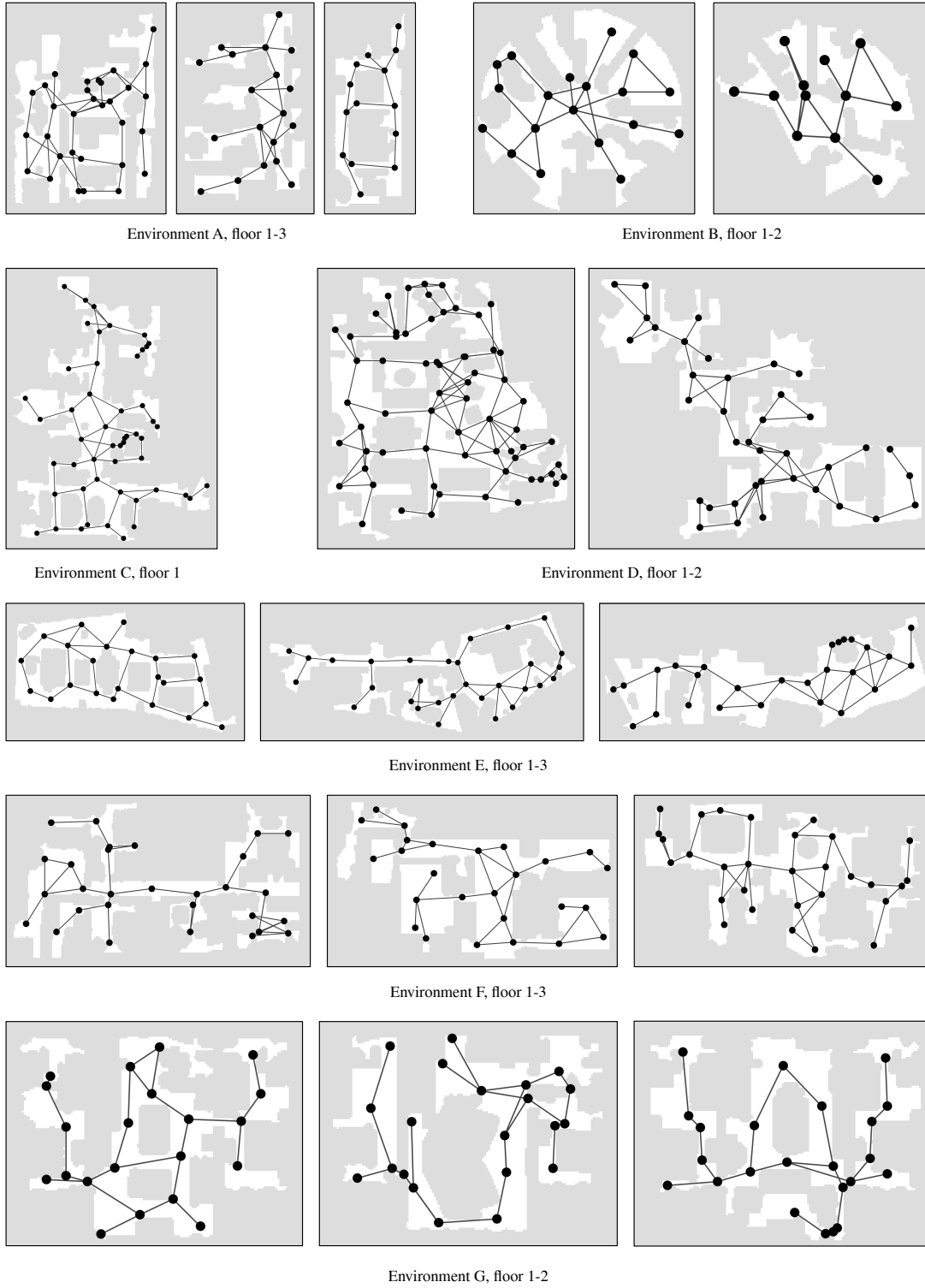


Figure 2: **Agent-generated topological maps.** The graphs are reasonably sparse but still cover a large portion of the traversable space. Certain nodes may be difficult to reach or close to walls, such as in Environment B floor 1.

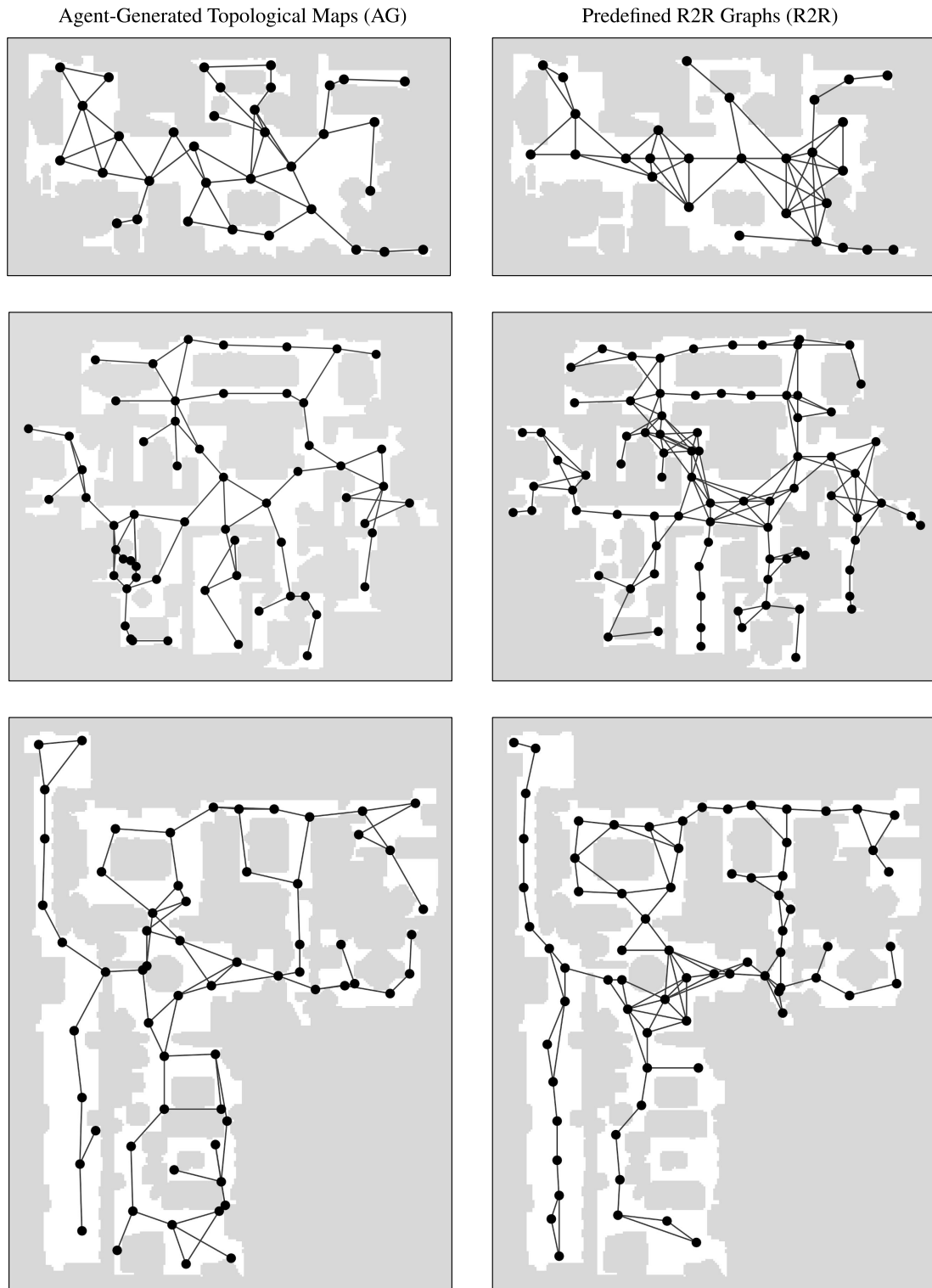


Figure 3: **Comparison of agent-generated maps (left) and R2R predefined graphs (right).** In general, the agent-generated maps are sparser, whereas the R2R graphs may have dense clusters of nodes.

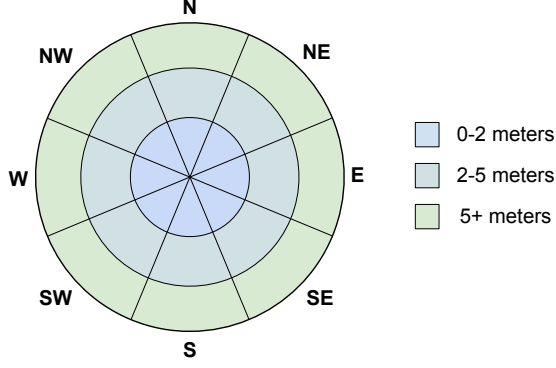


Figure 4: Directed edge labels for the topological map are represented as discretized polar coordinates. The chart is represented as a top-down view with the agent in the center, and orientations represented as compass directions.

1.3. Topological Map Representation

As stated in the main paper, each topological map node is encoded as a ResNet152 feature [5], and each edge is mapped to an embedding corresponding with the edge category. The edge categories are in discretized polar coordinates which are visualized in Fig. 4.

2. Method

2.1. Cross-Modal Planning

2.1.1 Graph Neural Network (GNN)

Our graph neural network consists of 6 sequential graph network (GN) blocks. Borrowing the notation from Battaglia *et al.* [3], each GN block is comprised of update functions $\phi^v(\cdot)$, $\phi^e(\cdot)$, $\phi^u(\cdot)$ and aggregation functions $\rho^{e \rightarrow v}(\cdot)$, $\rho^{e \rightarrow u}(\cdot)$, $\rho^{v \rightarrow u}(\cdot)$.

Update functions. We implement all update functions $\phi(\cdot)$ as multi-layer perceptrons (MLPs). Each MLP has 4 fully-connected layers of hidden dimension 1024, each followed by batch normalization [6] and ReLU except the last layer.

Aggregation functions. We implement all aggregation functions $\rho(\cdot)$ as element-wise summations.

Inputs to first GN block. The first GN block the same dimensions as the intermediate GN blocks. As input to the first GN block, we use the following:

- Node features $\mathbf{v}_i^{(1)} \in \mathbb{R}^{1024}$ for the i th node in the topological map. Each node embedding is a ResNet152 embedding.

- Edge features $\mathbf{e}_j^{(1)} \in \mathbb{R}^{256}$ for the j th edge in the topological map. Each edge embedding is a learned feature for the corresponding edge category (Fig. 4).
- A single global feature $\mathbf{u}^{(1)} \in \mathbb{R}^{512}$ which is a learned embedding.

Intermediate GN blocks. The intermediate GN blocks use the following input and output dimensions for intermediate layer k :

- Node features $\mathbf{v}_i^{(k)} \in \mathbb{R}^{1024}$ for the i th node in the topological map.
- Edge features $\mathbf{e}_j^{(k)} \in \mathbb{R}^{512}$ for the j th edge in the topological map.
- A single global feature $\mathbf{u}^{(k)} \in \mathbb{R}^{512}$.

Outputs of last GN block. The last GN block (index m) uses the same input dimension as the intermediate GN blocks. For the output dimensions, we use the following:

- Node features $\mathbf{v}_i^{(m)} \in \mathbb{R}^{768}$ for the i th node in the topological map.

2.1.2 Cross-Modal Transformer

Architecture. Our cross-modal transformer uses a similar architecture to LXMERT [13]. While LXMERT uses object-level image embeddings which are encoded with object position information via extra layers, we use node embeddings (from the GNN) and encode position using learnable position embeddings as described in the main paper.

For the language encoding branch, we map each word to a learnable embedding and each index position to a learnable embedding as well. For words w_i at index i of a navigation instruction, we encode the instructions as:

$$h_i = \text{LayerNorm}(\text{WordEmbed}(w_i) + \text{IdxEmbed}(i)) \quad (3)$$

Single-modality encoders. Each layer of the single-modality encoders contains a self-attention sub-layer followed by a feed-forward sub-layer. Each sub-layer is followed by a residual connection [5] and layer normalization [2].

Cross-modality encoders. Each layer of the cross-modality encoder contains a cross-attention sub-layer, a self-attention sub-layer, and a feed-forward sub-layer. Each sub-layer is followed by a residual connection [5] and layer normalization [2].

Number of layers. In total, we use 9 single-modal language encoding layers, 5 single-modal map encoder layers, and 5 cross-modal layers. For further details, we refer the reader to Tan and Bansal [13].

Classification head. We use a linear layer in the classification head to map from the 768-dimensional transformer node outputs to the classification logits.

Loss function. We use a standard cross-entropy loss for training the cross-modal transformer planner (CMTP), where classification is done over the nodes and [STOP] action. The same loss is used for CMTP-Repeated (CMTP-R) except the classification is done only over the nodes. For CMTP Binary Cross Entropy (CMTP-BCE), we use the following loss function:

$$L = 0.5CE(x_{node}, y_{node}) + 0.5BCE(x_{stop}, y_{stop}) \quad (4)$$

where CE represents cross-entropy loss over the node predictions x_{node} and ground truth y_{node} , and BCE is binary cross entropy over the predicted stop action x_{stop} and ground truth y_{stop} .

Training details. For the training the planner, we use the AdamW optimizer [7, 9] with a learning rate of $2e-5$ and a linear warm-up schedule. We use a weight decay of 0.01 on all non-bias and layer normalization [2] weights. We use a batch size of 16. All planners (GNN baseline and CMTP models) are trained from scratch without pre-training.

2.2. Controller

2.2.1 Logic

The controller is given a navigation plan, in the form of a topological path, and executes the plan in the simulated environment by making observations and outputting an action at each time step. Specifically, Alg. 1 describes the logic in which the controller interacts with the environment.

The controller first retrieves the RGBD panoramic observation for each node in the navigation plan. At the beginning of an episode, the controller assumes that the agent is in proximity to the first node o_1 in the plan. It then selects the o_2 as the subgoal. At each time step, the agent makes an observation of the environment o_{curr} , and uses an odometry sensor reading of the agent’s orientation to rotate o_{curr} to align with the orientation of o_1 and o_2 .

C_{high} predicts a waypoint based on these three panoramas, a bias function B is applied to the prediction to alleviate perceptual aliasing, and then C_{low} translates that waypoint into a robotic action. If the predicted waypoint is within a hyperparameterized proximity threshold such that $\rho \leq d$, the agent consumes the current subgoal, and the next

Algorithm 1: Controller Logic

```

input :  $\{o_1, o_2, \dots, o_n\}$  panoramas at planned nodes
param:  $\sigma^2$  variance of bias function
          $d$  localization threshold
 $\phi'_0 \leftarrow \vec{1}, k \leftarrow 1, t \leftarrow 1$ , OBSERVE  $o_0$ 
while  $k \leq n$  do
    OBSERVE  $o_{curr}$ 
     $\phi_t, \rho_t \leftarrow C_{high}(o_{k-1}, o_{curr}, o_k)$ 
    if  $\rho_t \leq d$  then
         $k \leftarrow k + 1$ 
    else
         $\phi'_t \leftarrow B(\phi_t, \phi'_{t-1}; \sigma^2)$ 
        EXECUTE  $C_{low}(\arg \max_i \phi'_{t_i})$ 
         $t \leftarrow t + 1$ 
    end
end

```

node in the plan becomes the new subgoal. This process is repeated until there are no more nodes in the plan.

2.2.2 High level controller (C_{high})

The high level controller, similar to [10], is a robot-agnostic function that predicts a waypoint when given an agent state. As specified above, the agent state is denote by an RGBD panoramic observation triplet $\langle o_{prev}, o_{curr}, o_{subgoal} \rangle$. The output of the high level controller is a waypoint, towards which the agent should move to in order to reach the position of $o_{subgoal}$. This waypoint is denoted in a discretized polar coordinate system defined by $\langle \phi, \rho \rangle$, where ϕ is the index of one of 24 partitions of each spanning 15 degree, and ρ is a numeric distance.

As such, the high level controller is a function that maps the agent state of observations to a waypoint:

$$C_{high}(o_{prev}, o_{curr}, o_{subgoal}) \rightarrow \hat{\phi}, \hat{\rho} \quad (5)$$

Architecture. As seen in Figure 5, our controller architecture is a general 2-staged approach. We share the same convolutional encoder to extract embeddings from each of our observations O_{prev} , O_{curr} and O_k as explained in the paper. These embeddings are then passed to the classifier and regressor. Note that the input to the classifier is a concatenated 5-tuple of $(z_{prev}, z_{prev} - z_{curr}, z_{curr}, z_{curr} - z_k, z_k)$, and the input to the regressor is a concatenated 3-tuple of $(z_{curr}, z_{curr} - z_k, z_k)$, where each z is the 3584 dimensional output from the convolutional encoder. We find that including these “deltas” that are the difference between the embeddings is helpful for training a better controller. The architectures for the convolutional encoder, classifier, and regressor are depicted in Table 1, Table 2, and Table 3, respectively.

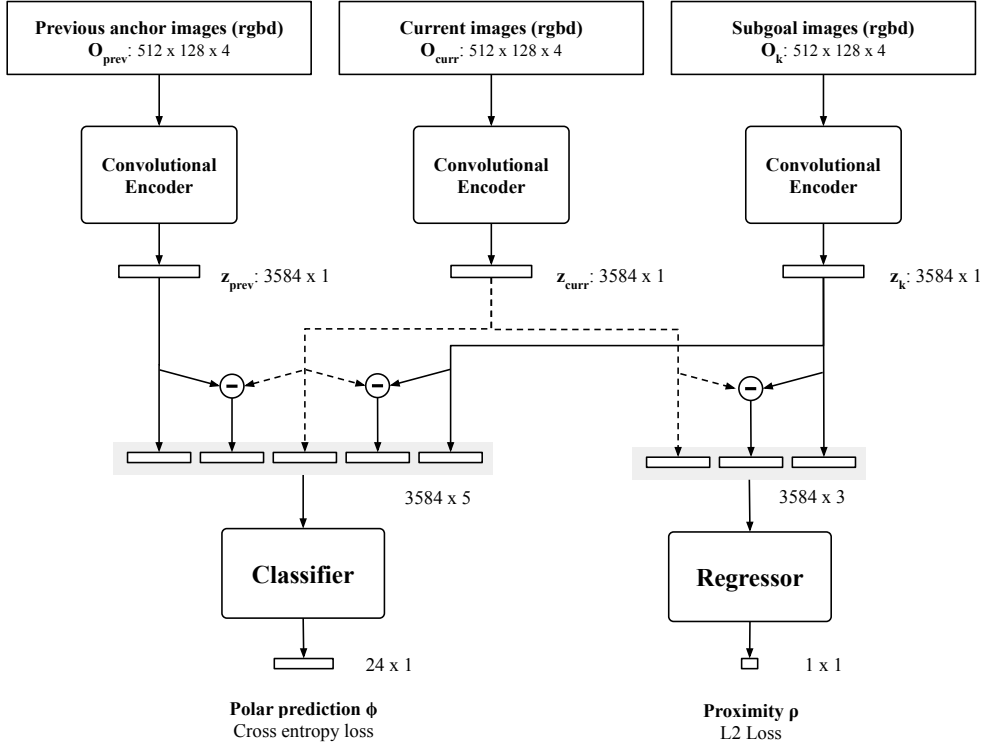


Figure 5: **High-level controller architecture.** The high-level controller is composed of three main components: the convolutional encoder, classifier, and regressor. See the text for details.

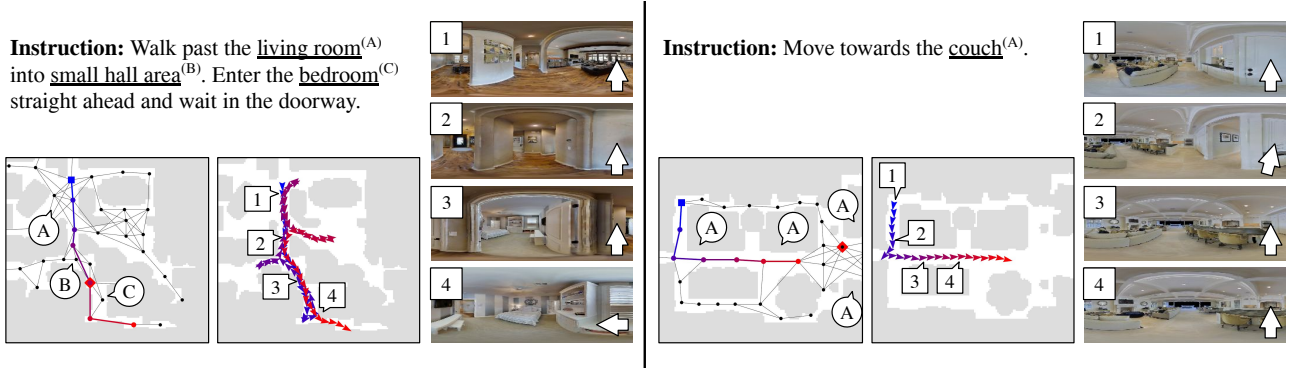


Figure 6: **Failure cases.** On the left, the planner overshoots the ground truth destination, failing to stop at the right time. The controller successfully navigates into the room and accidentally backtracks out. However, it still reaches the planner destination at the end of the episode. On the right, the navigation instruction is too ambiguous. There are many couches in the environment and it is unclear which couch to move towards. See the text for more details.

Training. We perform the training of the high level controller network offline using a synthetically generated dataset of 2 million data samples. Each data sample consists of $\langle o_{prev}, o_{curr}, o_{subgoal} \rangle$ and its label $\langle \phi, \rho \rangle$. Both output values are simultaneously optimized, with the direction index ϕ optimized by cross-entropy (CE), and dis-

tance ρ by mean squared error (MSE):

$$L = 0.5CE(\phi, \hat{\phi}) + 0.5MSE(\rho, \hat{\rho}) \quad (6)$$

During training, we also perform data augmentation by randomly rotating all observation panoramas and ρ by a same random amount.

Layer	In	Out	Kernel	Stride
PConv	4	64	7	3
Batch Norm	64	64		
ReLU	64	64		
Max Pool	64	64	3	2
PConv	64	128	5	2
Batch Norm	128	128		
ReLU	128	128		
Max Pool	128	128	2	2
PConv	128	256	3	1
Batch Norm	256	256		
ReLU	256	256		
Max Pool	256	256	2	2
PConv	256	512	3	1
Batch Norm	512	512		
ReLU	512	512		
Max Pool	512	512	2	2
Flatten+FC		3584		

Table 1: **Convolutional encoder architecture.** Each row represents a layer in the network along with the number of input channels (In) and number of output channels (Out).

Layer	Shape In	Shape Out
FC	3584 x 5	512
ReLU	512	512
FC	512	256
ReLU	256	256
FC	256	128
ReLU	128	128
FC	128	24

Table 2: **Classifier architecture.** The classifier predicts the direction of the waypoint. See the text for details.

Layer Type	shape in	shape out
FC	3584 x 3	256
ReLU	256	256
FC	256	128
ReLU	128	128
FC	128	1

Table 3: **Regressor architecture.** The regressor predicts the waypoint distance from the agent.

Expert Waypoint. The expert waypoint is produced by ground truth positions on a traversability map (a grid). We first build a traversal graph G of this grid, with edges connecting neighboring positions weighted by their L2 distance plus a penalty if close to a wall. Then, given any agent position p_{curr} and a subgoal position $p_{subgoal}$, we can calculate a lowest cost path $p = \{p_1, p_2, \dots, p_n\}$, where $p_1 = p_{curr}$

and $p_n = p_{subgoal}$.

To get an expert waypoint, we define a maximum look-ahead distance m , and we take the position p_k such that travelling from p_1 to p_k via the path is as far as possible, but no farther than m . Note that in this setting, when the agent position is sufficiently close to the subgoal, $p_k = p_{subgoal}$ is chosen such that the expert waypoint position is the subgoal position.

C_{high} is a simple function that produces a waypoint based on observations, which can be trained completely offline given a sufficiently large and robust dataset of samples. We also use DAgger [11] to further improve on the initially collected dataset.

2.2.3 Low level controller (C_{low})

We use a simple low-level controller C_{low} that rotates the agent to orient towards the predicted waypoint and walk in a straight line towards it. Because of the naive nature of this low-level controller, the waypoint predicted by C_{high} must be highly accurate in order to achieve good performance. Our experiments in the main paper indicate that despite using a simple controller, the agent is still able to successfully follow navigation plans with over 80% success rate. This performance could be further improved by using more sophisticated low-level controllers without requiring substantial changes to the rest of the overall approach due to our modular setup.

3. Experiments

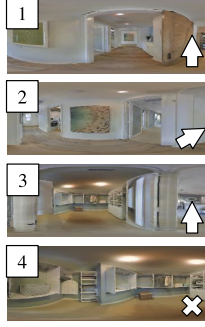
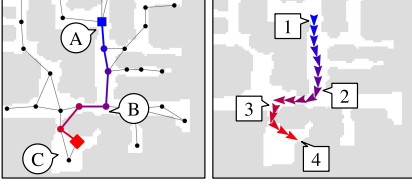
We use the iGibson simulator [14, 15] along with the VLN-CE data [8] for our experiments. To perform training and evaluation, we convert the VLN-CE trajectories into paths in our agent-generated (AG) maps and pre-defined Room2Room (R2R) graphs [1]. For each episode, we compared the ground truth 3D trajectory positions with the closest nodes in appropriate topological maps.

As a function of our data generation process, a small portion of the VLN-CE episodes were discarded for both train and validation splits. This may occur, for instance, if the ground truth start/end goal positions do not have any nearby nodes in the topological map. In this scenario, the episode is not used for training or validation.

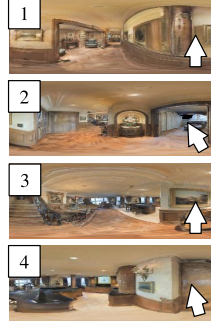
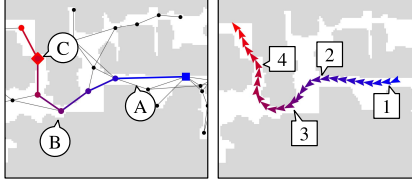
Differing VLN-CE performance from original paper.

We trained the base VLN-CE model (no PM, DA, Aug. defined in [8]) in a different simulator (iGibson), resulting in different numbers. As the VLN-CE authors note, their model performance varies even with different versions of the *same simulator* (Habitat [12]). Moreover, a small number of dataset episodes were discarded for training/evaluation as noted above.

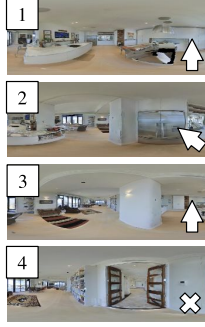
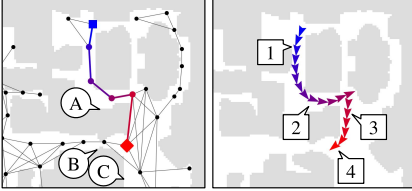
Instruction: Exit room through barn style door^(A) and head straight down hallway towards the painting on the wall^(B). Turn right into the bedroom. Take a hard left through the mirrored door. Turn slight left and wait by the white floor length shelves^(C).



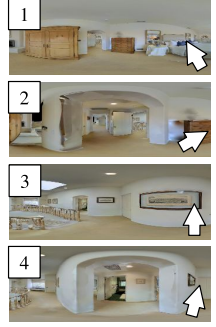
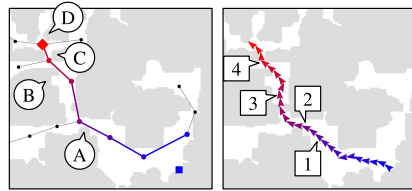
Instruction: Turn right and walk along the hallway^(A). Turn slightly left and walk towards the stairs^(B). Turn slightly right and walk towards the kitchen area. Wait by the painting in the kitchen area^(C).



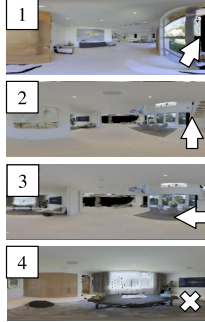
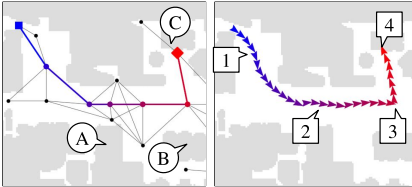
Instruction: Walk towards the fridge^(A). Turn left and walk around the corner to the right. Stop in front of the doorway^(B) before reaching the bookshelf^(C).



Instruction: Exit the bedroom through the archway^(A), turn right, pass the stairs^(B) and enter the archway^(C) on the right. Stop in the hallway and wait near the bathroom door^(D).



Instruction: Enter the building and walk past the stairs^(A). Turn hard left at the glass table^(B) and go into the room. Stop in front of the wooden table^(C).



Instruction: Walk over the star tile^(A) into the bedroom and walk towards the bed^(B). Walk around the bed, and stop next to the left set of windows^(C).

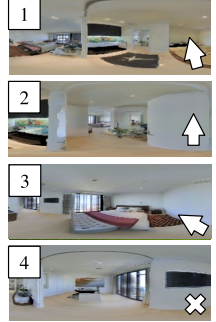
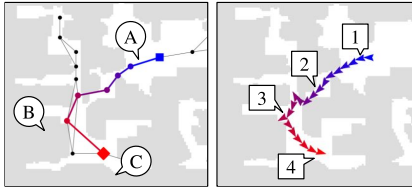


Figure 7: **Integrated system VLN examples.** We show 6 success cases of our CMTP approach. The agent successfully predicts plans according to the directions and landmarks specified in the instructions, while the controller executes the plans via low-level actions.

3.1. VLN Qualitative Results

We provide more qualitative results of our integrated system (CMTP) in Fig. 7. As can be seen in the figure, the planner successfully generates plans which land near the ground truth goal position. Many of the trajectories involve going into multiple rooms or navigating past landmarks specified in the instructions.

We also show example failures and inefficiencies in Fig. 6. On the left side of Fig. 6, we see the planner traverses past the ground truth goal position, failing to stop at the right time. On the other hand, the controller initially successfully navigates into the bedroom. However, it mis-

takenly backtracks out of the room and after numerous self-corrections, finally correctly navigates into the bedroom and stops at the end of the navigation plan.

In the example on the right of Fig. 6, the provided instruction is too ambiguous. The instruction tells the agent to move towards the couch, but there are several couches in the environment and it is unclear which couch is being referred to in the instruction. As a result, the planner predicts a plan that moves towards a couch and the controller successfully executes the navigation plan, but this is not the same couch as specified in the instruction.

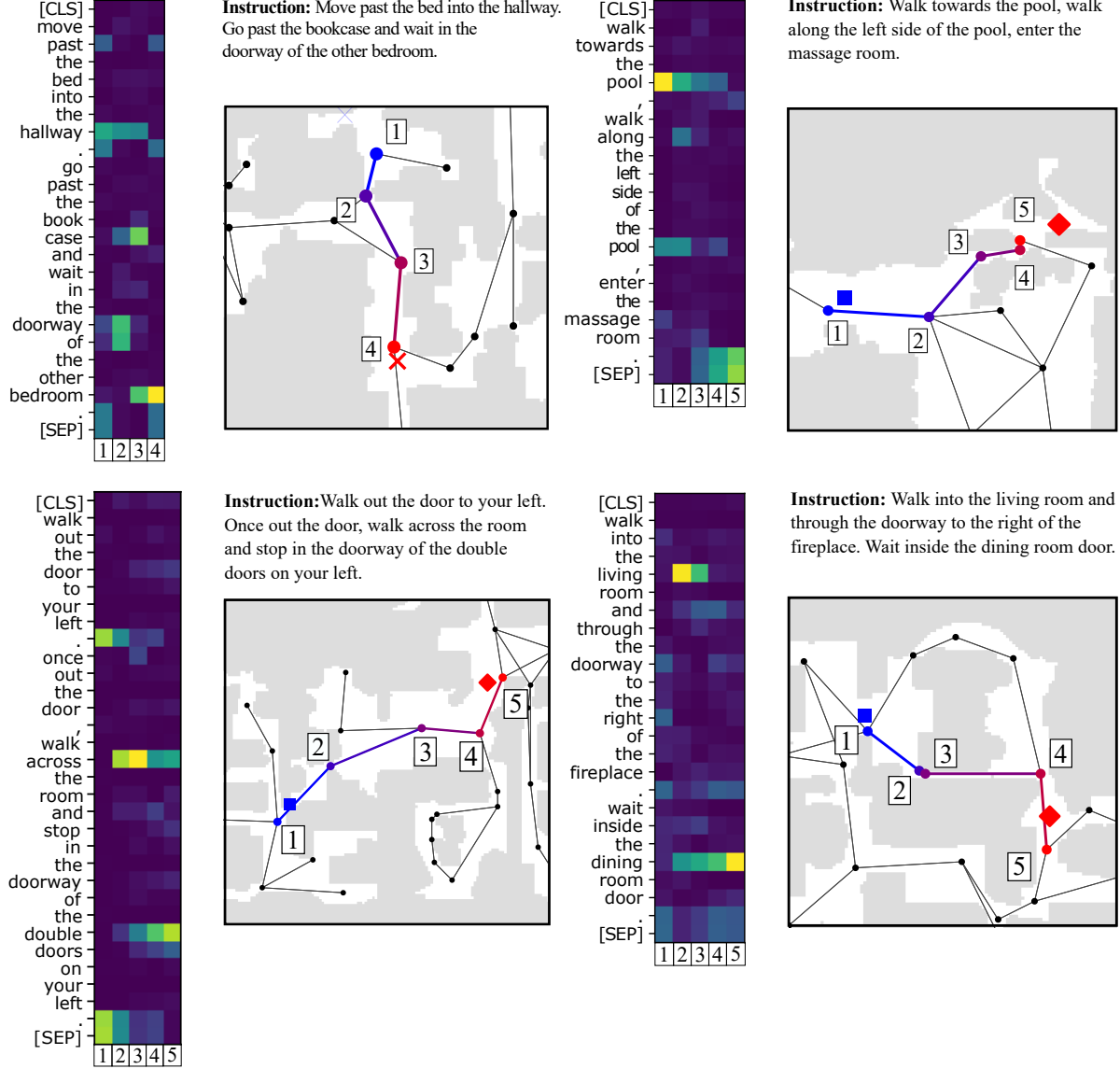


Figure 8: **Cross modal attention.** These visualizations follow the same format as the main paper. See the text for details.

3.2. Cross Modal Attention

In Fig. 8 we include additional examples of cross modal attention. These follow the same structure as the example in the main paper. In the top-left example, we see “hallway” associated with nodes in the hallway, and the “bedroom” word associated with the bedroom node (node 4).

Similarly, in the bottom-left example, we see that the word “across” has high correlation with the nodes involved in traversing across the room (nodes 2, 3, 4), while “double doors” is related to the last node. These results suggest that the model is capable of aligning instruction words with spatial locations.

References

- [1] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. 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, 2018. 1, 8
- [2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 5, 6
- [3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 5
- [4] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *International Conference on Learning Representations*, 2019. 1
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5
- [6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 5
- [7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6
- [8] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. 2020. 8
- [9] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. 2018. 6
- [10] Xiangyun Meng, Nathan Ratliff, Yu Xiang, and Dieter Fox. Scaling local control to large-scale topological navigation. *arXiv preprint arXiv:1909.12329*, 2019. 1, 6
- [11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 8
- [12] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9339–9347, 2019. 8
- [13] Hao Tan and Mohit Bansal. LXMERT: Learning cross-modality encoder representations from transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5100–5111, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. 5, 6
- [14] Fei Xia, Chengshu Li, Kevin Chen, William B Shen, Roberto Martín-Martín, Noriaki Hirose, Amir R Zamir, and Li Fei-Fei Silvio Savarese. Gibson env v2: Embodied simulation environments for interactive navigation. 2019. 8
- [15] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Michael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibbon benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020. 8
- [16] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. Towards New Computational Principles for Robotics and Automation*, pages 146–151. IEEE, 1997. 1