Tree-Structured Trajectory Encoding for Vision-and-Language Navigation

Xinzhe Zhou[1], **Yadong Mu**[*] [1,2]

[1] Wangxuan Institute of Computer Technology, Peking University
[2] Peng Cheng Laboratory
{zhouxinzhe1023, myd}@pku.edu.cn

Abstract

Over the past few years, the research on vision-and-language navigation (VLN) has made tremendous progress. Many previous works attempted to improve the performance from different aspects like training strategy, data augmentation, pre-training, etc. This work focuses on a rarely-explored aspect in VLN, namely the trajectory organization and encoding during the navigation. Most of existing state-of-the-art VLN models adopt a vanilla sequential strategy for encoding the trajectories. Such strategy takes the whole trajectory as a single sequence to estimate the current state, no matter whether the agent moved smoothly or perhaps made mistakes and backtracked in the past. We show that the sequential encoding may largely lose this kind of fine-grained structure in the trajectory, which could hamper the later state estimation and decision making. In order to solve this problem, this work proposes a novel tree-structured trajectory encoding strategy. The whole trajectory is organized as a tree rooted from the starting position, and encoded using our Tree-Transformer module to fully extract the fine-grained historical information. Besides, as the spatial topology could be easily embedded in the trajectory tree, we further design a tree-based action space to allow the agent making long-range error-correction in one decision. We implement the holistic agent based on cross-modal transformer and train it with a newly-proposed Tree-nDTW reward. On the benchmark dataset R2R, our model achieves a surpassing success rate (SR) of 68% on `val-unseen` and 66% on `test`. We further conduct extensive ablation studies and analyses to provide more insights for the effectiveness our designs.

Introduction

Vision-and-Language Navigation (VLN) is a growing field that combines vision-and-language learning with embodied decision process (Anderson et al. 2018b). The task requires an embodied agent (robot) to move along an expected path specified by a natural language instruction in some environment. Generally, the agent should understand the instruction and perceive its surroundings through a first-person RGB camera, and then sequentially decides its next move. The potential applications include domestic service, emergency rescue, etc.

*Corresponding author.

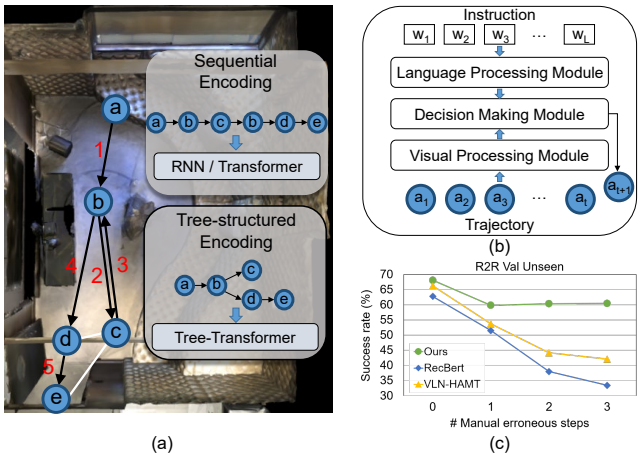


Figure 1: (a) An illustrative example comparing previous sequential strategy with our tree-structured strategy for encoding trajectory. (b) The structure of a typical VLN agent today. (c) The results of manually imposing error-and-corrections of various steps at the beginning for different models.

Since its proposal, the field of VLN has witnessed continuous progress from different aspects (Anderson et al. 2018b; Fried et al. 2018; Wang et al. 2019; Hong et al. 2021; Chen et al. 2021). Fig. 1(b) depicts the structure of a typical VLN model today. It mainly consists of three parts: the Language Processing Module processes the instruction, the Visual Processing Module encodes the historical and current visual observations (a.k.a., the trajectory), and the Decision Making Module aggregates the two modalities for the final decision (i.e., selecting the next navigable position). Although the model detail varies in previous works, we notice that their trajectory encoding strategies have mostly followed a similar paradigm–to encode the whole trajectory as a single sequence using sequential models like LSTM (Hochreiter and Schmidhuber 1997) or transformer (Vaswani et al. 2017). Some exceptions like EGP (Deng, Narasimhan, and Russakovsky 2020) or SSM (Wang et al. 2021) maintain a graphical representation of the visited environments, but they still resort to sequentially encoding the trajectory to estimate the current state.

The sequential encoding has the merit of keeping the order of all historical steps, but they may lose the fine-grained structure in the trajectory. The structure here refers to the error-and-corrections[1] occurred during the navigation. Fig. 1(a) gives an illustrative example. At the 2-nd step, the agent went to "c"; then at the 3-rd step, it realized that the last action was incorrect, so it returned back to "b" and moved to "d" for correction. Such one-step error-and-correction is the most simplified case, and in more challenging scenarios, the mistake could last for several steps before the agent realizes and corrects it. Due to the potential language and visual ambiguity, this kind of error-and-correction could happen a lot. For example, for the state-of-the-art RecBert (Hong et al. 2021) and VLN-HAMT (Chen et al. 2021), about 10% samples in R2R `val-unseen` set involve at least one error-and-correction, despite the paths only take 4-6 steps. These error-and-corrections represent the failed attempts that the agents made in the past, so the involved steps should not be mixed with ordinary steps when encoding the trajectory. On the other hand, it may be sub-optimal to directly discard the erroneous steps as they could also be useful for later navigation. For example, they could help clarify the confusion, broaden the agent view, enrich the environment topology, etc.

Based on the above analysis, we believe it is important to reflect the fine-grained structure when encoding the trajectory. However, the vanilla sequential encoding strategy may lack such ability. To provide an evidence, we manually control several state-of-the-art VLN agents and ours to conduct a k-step error-and-correction (move in an erroneous way for k steps and return back reversely) before letting them move freely. This introduces $2k$ error-and-correction steps in their trajectories, and if their encoding successfully identifies such structure, the agents should be aware of the mistake and the performance should not be downgraded greatly. However, as shown in Fig. 1(c), all agents but ours show a huge drop in SR. Besides, as the erroneous path grows longer, other models perform even worse but ours could maintain a stably high performance. This indicates that those models may have not correctly interpreted the error-and-correction steps, which leads to the abnormal performance.

Therefore, in this work, in order trying to solve the above problem, we propose a novel tree-structured trajectory encoding strategy. The core idea is to organize the whole trajectory as a tree instead of a plain sequence. Our insight is that we realize the error-and-correction happened at some step t could be characterized by a branching in the t-th position of the trajectory–one branch records the erroneous steps and the other branch follows later steps after the error is corrected. The branching structure reflects the characteristics of the erroneous steps, *e.g.*, they represent a failed attempt which is parallel with later attempts. Besides, as error-correction (backtracking) is to move reversely along the erroneous steps, they need not to be recorded explicitly as long as the erroneous steps are kept in the branch. This ensures the edges could only go from earlier to later positions, thus forming a strict tree hierarchy. Fig. 1(a) provides an example of the tree and the corresponding sequential trajectory.

Based on the tree-structured trajectory, we further propose three improvements to better fit the VLN task. First, as the tree is structurally different from the sequence, we design a novel Tree-Transformer module to encode our trajectory trees. Different from works in natural language processing that hierarchically encode the (syntax) trees using constituent attention (Wang, Lee, and Chen 2019) or tree-convolution (Harer, Reale, and Chin 2019; Sun et al. 2020), we propose novel depth-embedding and tree attention to adapt vanilla transformer (Vaswani et al. 2017) to fit tree structures. Second, we extend the conventional one-step action space to a new tree-based action space by embedding the spatial topology into the trajectory tree, in which the step-by-step backtracking actions are replaced with forward moves from post-backtracking positions so that the agent could achieve long-range error-correction in one decision. Besides, the tree also facilitates us to keep track of the current pursuing path in separation with all error-and-corrections, which better represents the current attempt towards completing the instruction compared with the full trajectory that includes all erroneous steps. This inspires us to propose a new Tree-nDTW reward for training the agent that encourages it to focus on improving the current attempt during the navigation.

The main contributions of this work are summarized as follows:

- We analyzed the problem in current sequential trajectory encoding methods, and proposed a novel tree-structured encoding strategy. Based on the trajectory tree, we designed a Tree-Transformer module to encode the tree structure. We further proposed to extend the one-step action space to a tree-based action space, and a new Tree-nDTW reward for better training the agent.
- We implemented our agent based on the cross-modal transformer. In the benchmark dataset, R2R, the agent achieved a surpassing SR of 68% on `val-unseen` and 66% on `test`. We also conducted extensive ablation studies and analyses to further provide more insights for the effectiveness of the proposed designs.

Related Work

Vision-and-Language Navigation

Vision-and-language navigation has been developed for several years since (Anderson et al. 2018b). In (Anderson et al. 2018b), Anderson *et al.* collected the first benchmark dataset, Room-to-Room (R2R), based on the photo-realistic Matterport3D environment (Chang et al. 2017). After it, many efforts had been devoted to promote the model performance. Wang *et al.* (Wang et al. 2018) proposed to combine the model-free and model-based Reinforcement Learning (RL) policy. Fried *et al.* (Fried et al. 2018) designed a speaker model to generate augmented data for training the navigation model. Ma *et al.* (Ma et al. 2019a) showed that training on auxiliary task / loss together with the navigation could help regularize the model. Later works like

[1]We define "error-and-correction" to be moving forward and returning back reversely.

(Zhu et al. 2020a; Wang et al. 2020b; Wang, Wu, and Shen 2020; Liang et al. 2022) further developed better regularization signals. The model structure had also been evolving continuously. RCM (Wang et al. 2019) introduced cross-modal attention into the model which helped align the two modalities. Anderson *et al.* (Anderson et al. 2019) utilized Bayes filter (Thrun, Burgard, and Fox 2005) to reformulate the VLN task into a Bayesian state tracking problem. Another group of works tried to design heuristic mechanisms into the navigator like regret (Ma et al. 2019b), active exploration (Wang et al. 2020a), curriculum learning (Zhu et al. 2020b; Zhang et al. 2021), dual optimization (Dou and Peng 2022; Wang et al. 2022) etc. Some other attempts include utilizing fine-grained features (Qi et al. 2020; Hong et al. 2020), graphical planning (Deng, Narasimhan, and Russakovsky 2020; Wang et al. 2021; Chen et al. 2022), data augmentation (Tan, Yu, and Bansal 2019; Fu et al. 2020; Parvaneh et al. 2020; Li, Tan, and Bansal 2022), etc.

Recently, the success of cross-modal transformer and pretraining in other vision-and-language tasks (Sun et al. 2019; Lu et al. 2019; Li et al. 2019a, 2020a; Tan and Bansal 2019; Su et al. 2020; Chen et al. 2020; Lu et al. 2020; Li et al. 2020b) raised a new trend to apply similar ideas in VLN. Huang *et al.* (Huang et al. 2019) designed two pretraining tasks based on the VLN data and used them to warm-start the RCM model. Li *et al.* (Li et al. 2019b) proposed to replace the original GloVe (Pennington, Socher, and Manning 2014) word-embedding with pretrained Bert (Devlin et al. 2019) or GPT (Radford et al. 2018) embeddings and showed consistent improvement. Hao *et al.* (Hao et al. 2020) used cross-modal transformer to pretrain the visual and language representations which could be transferred to downstream navigators. VLN-Bert (Majumdar et al. 2020) was one of the earliest works to apply transformer directly to VLN, however in a *post hoc* manner to re-rank candidate routes. Later RecBert (Hong et al. 2021) and ORIST (Qi et al. 2021) introduced recurrence into transformer and operated online. Airbert (Guhur et al. 2021) collected a large in-domain dataset, BnB, to pretrain the transformer-based models. VLN-HAMT (Chen et al. 2021) showed that a hierarchical transformer-based model could achieve surprisingly effective results after proper pretraining. HOP (Qiao et al. 2022) designed history-and-order aware proxy tasks to better pretrain VLN models.

Transformer for Tree-structured Data

There are only a few works in other fields involving transformers for tree-structured data (Wang, Lee, and Chen 2019; Harer, Reale, and Chin 2019; Sun et al. 2020). The work of (Harer, Reale, and Chin 2019) and (Sun et al. 2020) insert the Tree-Convolution operation into plain transformer for processing tree structures, while we make modifications directly to the transformer components and introduce less extra parameters. (Wang, Lee, and Chen 2019) propose the Constituent Attention mechanism to hierarchically merge natural language sequence to eventually form a tree, while our tree is formed manually and our tree transformer is for structurally encoding it, which renders their method not applicable.

Methods

Unless otherwise specified, we all assume that the agent is operated under the panoramic action space proposed by (Fried et al. 2018), *i.e.*, the agent moves by choosing a neighboring position provided by the discrete topology graph.

Tree-Structured Trajectory

In Fig. 1(a), we showed an example of the tree-structured trajectory. Here we provide a formal description about how to generate it. Initially, the agent is spawned at the starting position, denoted as P_0. We initialize the tree with a root node (depth $= 0$) termed N_0, and record the information about P_0 in N_0, like the visual features. As every node in the tree records a position in the environment, we could define a mapping f from an arbitrary node N to its corresponding position P, *i.e.*, $f(N) = P$. We additionally maintain a pointer to track the node that represents the agent current position.

Then, at every subsequent step t, assume the agent is situated at the position P_{s_t} with the current node $N_{s_t}(f(N_{s_t}) = P_{s_t})$. We first find the neighbors of N_{s_t} in the tree and group them as $\mathcal{N}_{tree}(N_{s_t})$, which consists of the parent and children nodes of N_{s_t}. With a little abuse of notation, we could define another set $\mathcal{N}_{tree}(P_{s_t}) = \{f(N)|N \in \mathcal{N}_{tree}(N_{s_t})\}$ to represent all neighboring positions of P_{s_t} that are already recorded in the neighborhood of $\mathcal{N}_{tree}(N_{s_t})$ in the tree. The agent would then make an action to decide its next move. Based on $\mathcal{N}_{tree}(P_{s_t})$, we divide all possible actions into three groups.

First, the agent moves to a neighboring position $P_{s_{t+1}}$ that is not included in $\mathcal{N}_{tree}(P_{s_t})$ ($P_{s_{t+1}} \notin \mathcal{N}_{tree}(P_{s_t})$), which means the next position corresponds to a new neighbor of N_{s_t}. In this case, we create a new node $N_{s_{t+1}}$ in the tree and associate $P_{s_{t+1}}$ with it. As the agent steps from P_{s_t} to $P_{s_{t+1}}$, $N_{s_{t+1}}$ is set to be a child of N_{s_t} to reflect the order. Its depth is therefore $N_{s_t}.depth + 1$. Besides, the pointer is changed from N_{s_t} to $N_{s_{t+1}}$ to track the agent movement.

Another type of action is when $P_{s_{t+1}} \in \mathcal{N}_{tree}(P_{s_t})$. This means the agent moves to a visited neighbor that is already recorded in the tree. Without loss of generality, we still name the corresponding node $N_{s_{t+1}}$. As $N_{s_{t+1}}$ already exists, we need not create any new node. Therefore, the only change in the tree is to update the pointer to $N_{s_{t+1}}$.

The last type of action is `stop`, where the whole navigation terminates and we simply mark P_{s_t} as the predicted target position. If we also care about the full trajectory, we could associate each node with a list to record all timestamps that the agent visits it. Then the full trajectory could be recovered afterwards.

Discussion. As stated before, the motive to design the tree-structured trajectory is to better characterize the error-and-correction information. After introducing the tree generation process, we now explain how to achieve this.

If the agent always moves forward (never stepping back), then all the actions belong to the first group above. The tree would only have one branch, which degenerates to a plain sequence. If once the agent decides to step back (backtrack) to correct the erroneous step(s), then according to the second

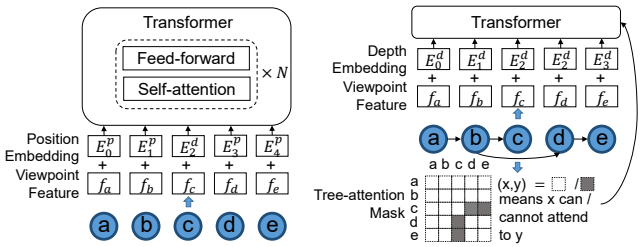


Figure 2: Comparison of the vanilla transformer for encoding sequences (left) and our Tree-Transformer to encode the tree structures (right).

rule, stepping back would not introduce any new node but only update the pointer from N_{cur} to the parent node N_{par}. Afterwards, the agent may move to another new neighbor of N_{par} to continue the navigation, which would drive the tree to grow a new branch from N_{par} and leave N_{cur} as a deprecated branch[2]. This is the case when one-step error-and-correction happens. In general, the agent could successively move back for k steps before continuing the navigation, which would result in a longer deprecated branch with k nodes.

Such deprecated branches are exactly how does the tree represent all the error-and-corrections. As a comparison, previous sequential trajectory would always append each step at last, no matter whether the agent is moving forward or backward. So in order to differentiate error-and-correction steps with ordinary steps, the agent is required to exhaustively compare every pair of adjacent steps to determine whether they are reverse with each other. The situation could get more complex if there are multi-step or nested error-and-corrections in the trajectory, which needs more sophisticated processing. The difficulty in differentiating the error-and-corrections in the sequential encoding may hamper the state estimation and decision making, *e.g.*, the agent may erroneously interpret error-and-correction steps as ordinary steps to mis-estimate the current progress.

Tree-Transformer

After forming the trajectory tree, we need to encode it so as to estimate the agent state at each step. Previous models like LSTM (Hochreiter and Schmidhuber 1997) or vanilla transformer (Vaswani et al. 2017) are not applicable as they are primarily designed for sequential data. Therefore, considering the effectiveness of transformer in encoding sequences and its flexibility, we propose two adaptations to the vanilla transformer to adapt it to be our Tree-Transformer module that fits tree structures.

Vanilla Transformer. As first proposed by (Vaswani et al. 2017) and depicted in Fig. 2(left), the vanilla transformer module is mainly composed of interleaved self-attention and feed-forward layers. To process sequential data, each element (feature vector) in the input sequence would often be added with a position embedding to represent the order. Then during the processing, every element would densely attend to all the elements in the sequence repeatedly to produce the final output.

Position Embedding → Depth Embedding. The main characteristic of the tree, compared to the sequence, is that it is a hierarchical structure. So it is important to represent the hierarchy in the encoding. The depth of each node is a natural choice to characterize the hierarchy. This is conceptually different from previous position embedding that orders all steps chronologically as we think the order within each attempt / branch is important while different attempts should be parallel with each other during the encoding. Therefore, as shown in Fig. 2(right), we replace the original position embedding with a new depth embedding. It is implemented by several learnable embedding vectors with each depth d associated with one specific vector.

Dense Attention → Tree-based Attention. The depth embedding encodes the hierarchy information in the tree, but it could not represent the detailed connection structure, *e.g.*, which node is the parent of a node at depth $d+1$ if there are multiple nodes at depth d. In order to remedy this, we further adapt the original dense attention into a tree-based attention operation. Specifically, as shown in Fig. 2(right), we introduce a mask in each self-attention layer to control the information flow. For every node, the mask only allows it to attend to all its ancestors and descendants in the tree, which is totally decided by the connection pattern. This guarantees that nodes in different branches could be explicitly differentiated during the encoding. Besides, as every node would share at least one common ancestor (the root) with any other node, it could still perceive and integrate the global information after stacked attention operations, which helps to produce a globally consistent encoding for decision making.

Tree-based Action Space

As stated before, in the tree-structured encoding, all backtracking steps need not to be recorded as the corresponding erroneous steps are already kept in the deprecated branches. In this section, we step further and propose a new action space that directly omits backtracking actions in the decision process.

Previous One-Step Action Space. In most previous methods, the agent decides its next action by choosing from all of its neighboring positions and moves one step, as shown in Fig. 3. This is reasonable if the agent is always moving forward following the instruction. However, if once the agent moves in a wrong way and could not continue forward, it needs to step back to recover from the mistake. The one-step action space may complicate the error-correction process as the agent needs to figure out the exact reverse move of its last action. The situation gets even worse if the error lasts for several steps, which requires the agent to learn to reverse earlier actions.

Tree-based Action Space. Based on the tree-structured trajectory above, we notice that the spatial topology could be naturally embedded in the tree, *e.g.*, upon adding a new node, we could record its relative position with the parent node in the tree. This facilities the agent to transfer between any pair of nodes in the tree without needing to de-

[2]A deprecated branch could be re-activated if the agent re-enters the associated nodes.

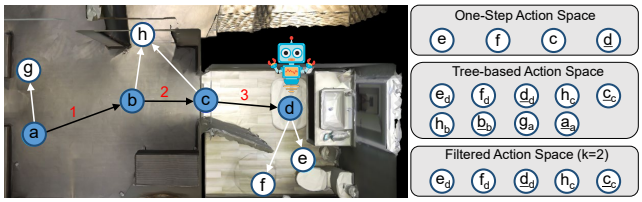


Figure 3: Example showing the difference of one-step action space, tree-based action space and its filtered version to keep only the recent 2 steps / nodes. Symbols with underline (*e.g.*, d) mean to stop at the corresponding nodes. Symbols with subscript (*e.g.*, e_d) mean to backtrack to the subscript node (d) and then step to the node associated with the symbol (e).

cide each intermediate move, which could easily accomplish long-range backtracking. Moreover, we realize that the reason why the agent backtracks is to exit the erroneous attempt in order to re-choose a way to move forward to. This inspires us to replace the previous one-step action space with the newly-proposed tree-based action space. The core idea is to replace the backtracking action at current node with forward moves from all post-backtracking nodes, *i.e.*, all nodes that the agent could reach through successive backtracking steps. In another word, we skip the backtracking process in the decision, and let the agent directly decide whether to continue the current attempt or re-start an alternative attempt.

Fig. 3 shows an illustrative example. The agent is currently situated at node d through the path $a-b-c-d$. In the one-step action space, all actions that the agent could choose are e, f, c, d, among which c is the backtracking step and d means to stop here. However in the tree-based action space, the backtracking step c is replaced with forward moves from $a, b,$ and c, which are the post-backtracking nodes. For example, h_b means the agent would backtrack to b and step to h. Note we exclude all executed actions from the candidates like c_b as they would lead to other post-backtracking nodes which is redundant. As can be seen, under the tree-based action space, the agent needs not to learn to conduct the backtracking itself. It only needs to determine the most promising forward move towards completing the instruction among all post-backtracking nodes and the current node. If x_y is chosen, the agent could be smoothly transferred to y along the recorded path, and then move to x. The tree-based action space is conceptually similar to the global action space proposed by (Deng, Narasimhan, and Russakovsky 2020; Wang et al. 2021). However, as we base the actions on the tree instead of the topological graph as in (Deng, Narasimhan, and Russakovsky 2020; Wang et al. 2021), each action in our model corresponds to exactly one attempt starting from the root node, while there could be multiple paths leading to the same action in the topology graph, which may confuse the agent.

Dynamic Filtering. Although the tree-based action space relieves the agent from learning to backtrack step-by-step, its drawback is also obvious–the candidate actions would continuously increase as the agent moves, which may hinder the efficiency and may introduce noise during decision making. In order trying to alleviate these problems, we further propose a dynamic filtering scheme. The insight comes from the observation that in most cases of backtracking, the agent would backtrack to nodes that it has recently visited. Put it another way, it is not likely the agent would backtrack to a far earlier node that it has not visited for a long time. For example, by adopting the full tree-based action space on R2R (Anderson et al. 2018b) `val-unseen` set, we observe that about 80% backtrackings are targeted at the nodes that have been visited in recent 5 steps. We also calculate the ratio for another backtracking-enhanced model SSM (Wang et al. 2021) and the result is also over 70%. We posit that this is because in most cases, the agent could realize its mistake timely, so that when backtracking, it only needs to backtrack to a recent node to correct it. Inspired by this, we propose to filter the original large action space to keep only the top-k recently visited nodes as candidate post-backtracking nodes. Fig. 3 shows an example of the case when $k = 2$. For implementation, we use a first-in-first-out queue of length k to conduct the filtering online. Throughout our experiments, we set $k = 5$ empirically. Later in experiments, we will show the detailed effect of varying the ks.

Tree-nDTW Reward

In addition to the above, another merit of the tree-structured trajectory is that it facilitates us to separate the current pursuing path from all previous error-and-corrections. Based on the tree, the current pursuing path is exactly the tree path from the root to the current node, and all previous error-and-corrections are deprecated branches grown alongside it. Compared with the full trajectory that is often used in previous models, this path better represents the current attempt towards completing the instruction. For example, if we use the conventional *normalized Dynamic Time Warping* (nDTW) (Ilharco et al. 2019) metric to evaluate the consistency with the ground truth path, the full trajectory would take all previous error-and-correction steps into account, which may affect or even dominate the result. Differently, the current pursuing path is separated with those abandoned attempts, so the result could purely reflect the quality of current attempt. This inspires us to improve the previous reward function that evaluates the full trajectory to a new reward evaluating the current pursuing path, so that the agent is encouraged to focus on improving its current attempt at each step. Besides, combined with the tree-based action space above, such reward also encourages the agent to compare the current pursuing path with other alternative paths after backtracking to determine whether it should proceed or backtrack, which better matches the target of the tree-based action space.

For implementation, we base our reward R on the popular nDTW (Ilharco et al. 2019) metric. To distinguish our reward with those that evaluate the nDTW of the full trajectory (Ilharco et al. 2019), we name ours the Tree-nDTW reward. The detailed equations are as Eqn. 1, where the $\mathrm{p}_t^{\mathrm{cur}}$ means the current pursuing path at step t, $\mathrm{nDTW}(\cdot)$ is the function to compute the nDTW, $\mathbb{I}\{\text{condition}\}$ is the indicator function that returns 1 if the condition is true, 0 otherwise. Following the convention, the agent is considered to

Models	R2R											
	val-seen				val-unseen				test			
	SR↑	NE↓	TL	SPL↑	**SR**↑	NE↓	TL	SPL↑	**SR**↑	NE↓	TL	SPL↑
Speaker-Follower	0.66	3.36	-	-	0.36	6.62	-	-	0.35	6.62	14.8	0.28
RCM	0.67	3.53	10.7	-	0.43	6.09	11.5	-	0.43	6.12	12.0	0.38
EnvDrop	0.62	3.99	11.0	0.59	0.52	5.22	10.7	0.48	0.51	5.23	11.7	0.47
AuxRN	0.70	3.33	-	0.67	0.55	5.28	-	0.50	0.55	5.15	-	0.51
Active Perception	0.70	3.20	19.7	0.52	0.58	4.36	20.6	0.40	0.60	4.33	21.6	0.41
SSM	0.71	3.10	14.7	0.62	0.62	4.32	20.7	0.45	0.61	4.57	20.4	0.46
RecBert	0.72	2.90	11.13	0.68	0.63	3.93	12.01	0.57	0.63	4.09	12.35	0.57
VLN-HAMT	**0.76**	**2.51**	11.15	**0.72**	0.66	**2.29**	11.46	**0.61**	0.65	3.93	12.27	**0.60**
Ours	0.75	2.86	14.07	0.68	**0.68**	3.49	17.31	0.57	**0.66**	**3.85**	19.62	0.55

Table 1: Results of different methods on the R2R (Anderson et al. 2018b) dataset. '-' means that the results are unavailable.

succeed if it stops within 3 meters of the goal (Anderson et al. 2018b).

$$\mathrm{R}(t) = \begin{cases} \mathrm{nDTW}(\mathrm{p}_t^{\mathrm{cur}}) - \mathrm{nDTW}(\mathrm{p}_{t-1}^{\mathrm{cur}}), & \text{if not stop,} \\ \mathbb{I}\{\text{the agent succeeds}\} + \mathrm{nDTW}(\mathrm{p}_t^{\mathrm{cur}}), & \text{if stop.} \end{cases} \tag{1}$$

The Holistic Agent

After elaborating the core components, we now describe the structure of the holistic agent. We adopt cross-modal transformer (*e.g.*, Lxmert (Tan and Bansal 2019)) as the main framework following recent practice (Majumdar et al. 2020; Hong et al. 2021; Qi et al. 2021; Chen et al. 2021). At every step t, the input consists of the instruction and the tree-structured trajectory. To integrate the tree-based action space, we append all the candidate views (representing the actions) to the corresponding nodes in the tree. As stated before, the backtracking action and the executed actions are excluded from the candidates. For processing, the candidate views at each node are added to the tree as virtual nodes as if these actions are already executed, *e.g.*, they are connected to the corresponding node as virtual leaf nodes. Intuitively, this helps the agent to encode each action in the context of the history. During processing, they are transformed together with the tree. The cross-modal transformer is mainly composed of interleaved intra-modality self-attention and inter-modality cross-attention layers. The Tree-Transformer is used to replace the self-attention layers for the visual modality. After the processing, we fuse the language representation (the CLS token) with each candidate view feature, and use a simple two-layer fully-connected network to predict a scalar score for this action. The decision is then made based on these scores. For more details, please refer to the supplementary material.

Experiments

Evaluation Metrics

Following previous conventions (Anderson et al. 2018b,a; Ilharco et al. 2019), we use several metrics to quantitatively evaluate the model performance: Success Rate (SR), Navigation Error (NE), Trajectory Length (TL), Success weighted by the normalized inverse of the Path Length (SPL), normalized Dynamic Time Warping (nDTW).

Results on R2R (Anderson et al. 2018b)

Tab. 1 shows the results on the R2R (Anderson et al. 2018b) dataset. As can be seen, our model achieves the best SR on the two unseen sets, especially `test` that is used by the online leaderboard[3], which demonstrates the effectiveness of our designs. The SR on `val-seen` is a bit lower than VLN-HAMT (Chen et al. 2021), however, as the environments in `val-seen` are fully perceived during training, the results could be inflated due to overfitting, *e.g.*, even earlier models like Speaker-Follower (Fried et al. 2018) could achieve nearly 70% SR. The less performance drop from seen to unseen environments in turn shows that our model is more overfitting-free and generalizes better.

Besides, we notice that the TL of our agent is higher than models like VLN-HAMT (Chen et al. 2021), although smaller than several others. The higher TL also causes lower SPL. There are mainly two reasons behind this. First, the tree-based action space allows the agent to conduct long-range transfer, which often involves many steps and increases the TL greatly. For reference, SSM (Wang et al. 2021) adopts a similar global action space and its TL is even longer than ours. Second, the Tree-nDTW reward encourages the agent to choose a more promising partial path (from root to leaf) with no penalty for the transfer between different partial paths, which helps the agent better follow the instruction but imposes less restriction on the TL. As comparison, the conventional reward functions evaluate either the distance to the goal or the nDTW of the whole trajectory, both of which would implicitly penalize the agent for moving too long. Although having reduced TL, one drawback of previous methods is that the agent may not be able to backtrack from errors and would try to approach the goal from wrong paths, which are greatly improved by our method as shown by later quantitative analyses and the qualitative examples.

Ablation Studies

Incremental Effect of Each Component. In order to illustrate the respective effect of each proposed component, we incrementally add each one of them to a base model. The

[3]https://eval.ai/web/challenges/challenge-page/97/leaderboard/270

Models	R2R val-unseen			
	SR↑	NE↓	TL	SPL↑
Baseline	0.59	4.43	13.97	0.53
+tree	0.61	4.07	13.02	0.54
+tree+action	0.63	4.06	14.90	0.54
+tree+reward	0.62	4.00	13.06	0.56
+tree+action+reward	0.68	3.49	17.31	0.57

Models	R2R val-unseen			
	SR↑	NE↓	TL	SPL↑
Ours	0.68	3.49	17.31	0.57
w/o depth-embedding	0.66	3.85	17.55	0.54
w/o tree-based attention	0.66	3.71	17.07	0.56
w/o both	0.63	4.07	20.21	0.54

Table 2: (top) Incrementally add each component to the base model. (down) Ablation study of the depth-embedding and tree-based attention in the Tree-Transformer module.

Rewards	R2R val-unseen		
	SR↑	TL	#BT
Tree-nDTW (Ours)	0.68	17.31	0.99
nDTW	0.63	15.12	0.65
Dist-to-goal	0.63	14.96	0.52
Both combined	0.63	14.90	0.54

Table 3: Ablation study of the reward.

results are shown in Tab. 2(top). Note as the tree-based action space and the Tree-nDTW reward are all based on the tree-structured encoding, they are both added after the tree-structured encoding. It can be seen that every component could indeed bring improvement, and with all the three combined, the model achieves the best result.

The Effect of the Tree-Transformer. We separately remove the depth-embedding, tree-based attention, and both of them in the Tree-Transformer module to see their impact for the tree-structured encoding. The tree-based action space and the Tree-nDTW reward are kept unchanged throughout the experiments. As shown in Tab. 2(down), both the depth-embedding and tree-based attention are indispensable for the Tree-Transformer encoding as removing either of them would harm the performance. If both of them are removed, the Tree-Transformer then degenerates to the vanilla transformer (Vaswani et al. 2017), where the performance shows a larger gap, demonstrating that it is not suitable to encode the tree structure. Besides, we also notice that the TL increases after both components are removed. We posit that this is because the vanilla transformer may confuse the agent about the trajectory structure, *e.g.*, the agent may mistakenly take a node as the parent of another node, which causes the agent to make erroneous backtracking decisions and increase the TL.

Effect of the Tree-based Action Space. As stated before, we use a queue of length k to dynamically filter the tree-based action space during decision making, and we empirically set $k = 5$ in the main experiments. Here we provide

Models	R2R val-unseen	
	Iden. nDTW↑	ΔnDTW↑
RecBert	0.67	0.047
VLN-HAMT	0.70	0.040
Ours	0.74	0.066

Table 4: The nDTW of the model-identified path and the average ΔnDTW before and after each error-correction.

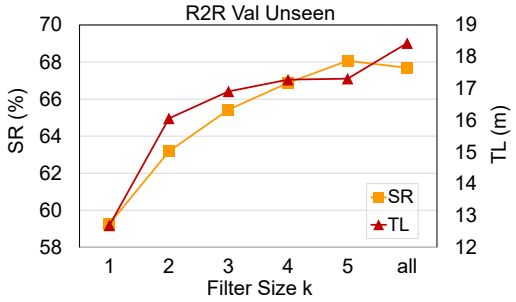


Figure 4: The impact of the k in dynamic filtering of the tree-based action space.

a list of results by varying the k from 1 to unlimited (all). Fig. 4 depicts the change of SR and TL with k. It could be seen that as the k grows, the SR first increases then decreases while the TL keeps increasing. This is because at first, when k is small, increasing k could provide more candidates for backtracking and helps the agent to correct its errors. However, as k grows large, the newly added candidates are less likely to be the proper node to backtrack to as they are earlier nodes that are not recently visited. The inclusion of these nodes instead brings more noise to the decision process, *i.e.*, there are more negative candidates which lower the agent confidence in making the correct decision. Differently, the TL keeps increasing as adding more earlier nodes would increase the average backtracking length.

Effect of the Tree-nDTW Reward. The Tree-nDTW reward enables the agent to focus on the current pursuing path which we believe is more suitable for training our model. In Tab. 3, we compare it with several other rewards that were adopted in previous works. The nDTW reward (Ilharco et al. 2019) evaluates the nDTW of the full trajectory. The dist-to-goal (Wang et al. 2021) rewards the agent for getting closer to the goal. And a combined reward (Hong et al. 2021; Chen et al. 2021) takes both metrics into consideration. It is clear that all the other rewards perform worse than our Tree-nDTW reward. Besides, the TLs of those rewards are consistently smaller than ours as they are all designed to improve the full trajectory, which however may hamper the agent to backtrack from erroneous steps. To illustrate this, we additionally count the average number of backtrackings (#BT) the agent has conducted in Tab. 3. It can be seen that Tree-nDTW indeed brings more backtrackings that helps the agent better correct its errors and achieve the best SR.

Models	RxR								
	val-unseen					test			
	SR↑	SPL↑	nDTW↑	SDTW↑	Iden. nDTW↑	**SR**↑	SPL↑	nDTW↑	SDTW↑
Monolingual baseline	28.5	-	44.5	23.1	-	25.9	-	41.5	21.0
Multilingual baseline	22.8	-	38.9	18.2	-	21.3	-	37.2	17.1
CLEAR-ResNet	-	-	-	-	-	34.7	31.6	50.5	29.6
CLIP-ViL	-	-	-	-	-	38.3	35.2	51.1	32.4
CLEAR-CLIP	-	-	-	-	-	40.3	36.6	53.7	34.9
VLN-HAMT(Multi)	56.5	**49.5**	**63.3**	**48.4**	64.7	53.1	**46.6**	**59.9**	**45.2**
Ours(Multi)	**58.7**	47.0	58.9	46.2	**67.3**	**54.1**	44.3	55.0	42.9

Table 5: Results of different methods on the RxR (Ku et al. 2020) dataset. SR, SPL, nDTW, and SDTW are shown in percentage (%). '-' means numbers unavailable.

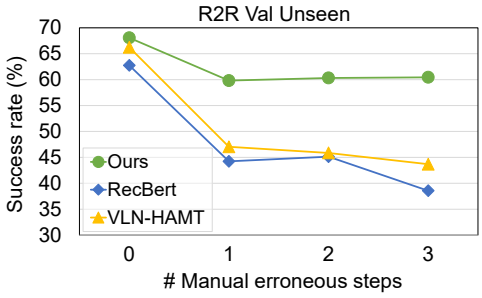


Figure 5: The detailed results of manually imposing erroneous steps at the beginning of different models.

Further Analyses

To further provide more insights about the proposed designs, we conduct additional analyses here.

Identifying the Expected Path. A major merit of our model compared with previous agents is that it does better in identifying the expected path described by the instruction. As the instruction in VLN has provided fine-grained steps to reach the goal, we believe it is equally important for the agent to identify the expected path as to reach the goal. To evaluate this ability, for the two state-of-the-art models, RecBert (Hong et al. 2021) and VLN-HAMT (Chen et al. 2021), and ours, we first remove all the error-and-correction steps from their final trajectories. The remaining path excludes all failed attempts and keeps only the most promising route towards completing the instruction, which could be seen as the final model-identified path corresponding to the instruction. We then compute the nDTW scores for these identified paths to see how well the model identifies the expected path. The results are shown in Tab. 4. As can be seen, our model shows a surpassing nDTW, indicating that it does better in identifying the correct path. We believe the superiority comes from both the tree-structured encoding that helps the agent to better distinguish the current pursuing path and the Tree-nDTW reward that encourages it to improve the current pursuing path towards completing the instruction.

Improved Error-Correction. Another merit of our model is that it is more effective in correcting its intermediate errors. To illustrate this, we dive deeper to see how much improvement each error-correction brings to the three models. Specifically, we compute the average ΔnDTW after each error-correction for each model. The results are also shown in Tab. 4. As can be seen, the average nDTW improvement of our model is also the largest, which shows that it does more effective error-correction thanks to the tree-based action space that relieves the agent from learning to backtrack step-by-step and the Tree-nDTW reward that allows the agent to better compare different attempts.

Besides, we also conduct a control experiment similar to Fig. 1(c) to test the agents' ability to correct the errors. Specifically, before letting the agents decide their actions, we manually control all three agents to move in an erroneous way for k steps but do not return them back, and see the impact of the erroneous steps on the final performance. The results are shown in Fig. 5. It could be seen that when faced with manual erroneous steps, our model again shows clear advantage, which further illustrates that it has better error-correction ability.

Results on RxR (Ku et al. 2020)

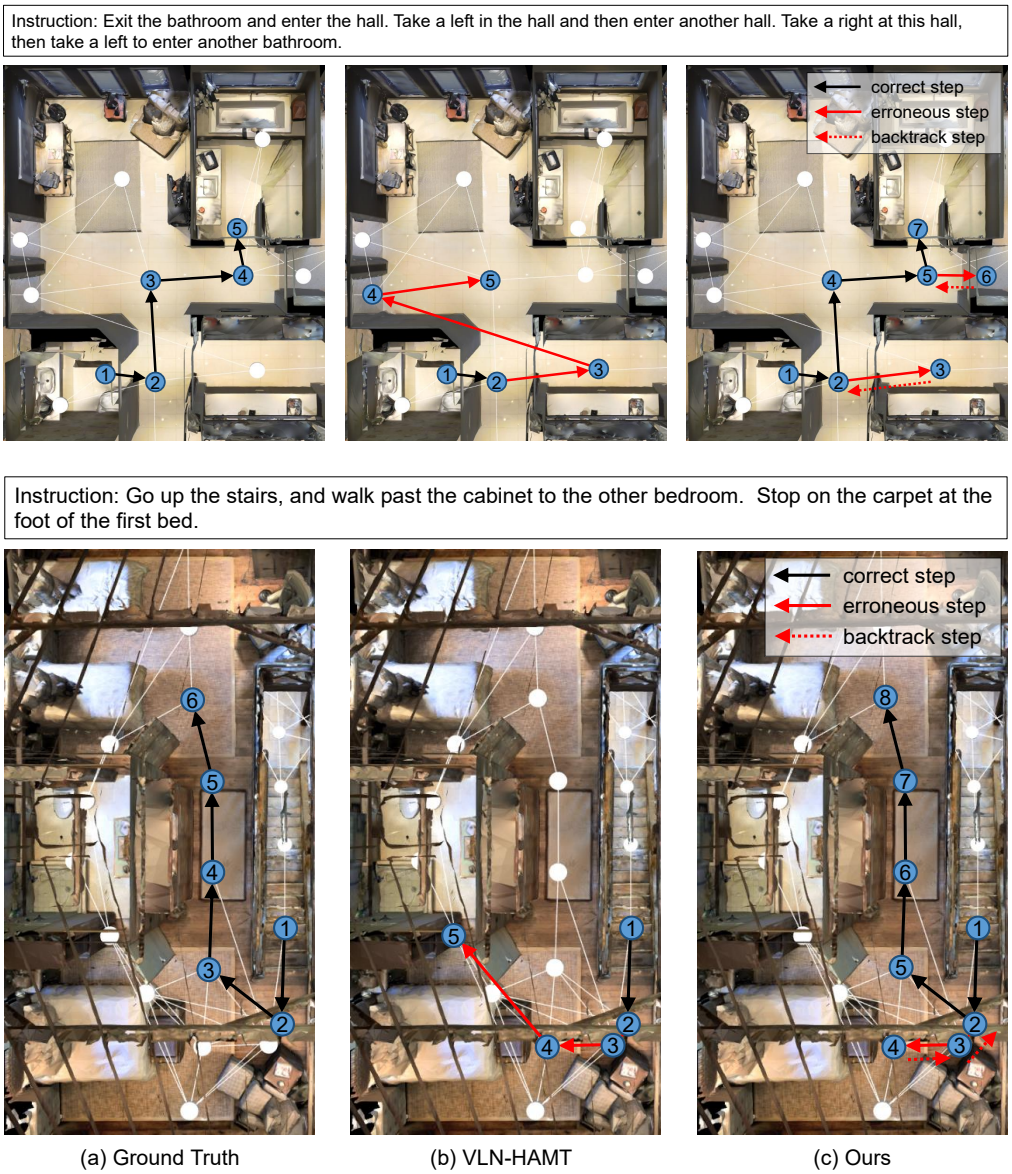
Besides R2R, We also tested our model on the more challenging RxR (Ku et al. 2020) dataset. The results on `val-unseen` and `test`[4] are shown in Tab. 5. As can be seen, our model still achieves the best SR on both sets, which is consistent with R2R (Anderson et al. 2018b). The SPL and nDTW are a bit lower than (Chen et al. 2021), due to the tree-based action space and the Tree-nDTW reward. However, the improved error-correction could help better identify the ground-truth path. To illustrate this, similar to above, we also compare the nDTW of the model-identified path on `val-unseen`. The annotations of `test` are unknown so could not be used for comparison. It can be seen that our model achieves better nDTW for identified path than VLN-HAMT (Chen et al. 2021), showing its better effectiveness in completing the instructions.

Qualitative Results

Besides the quantitative results, we also show in Fig. 6 several representative qualitative examples to illustrate the difference of our agent with VLN-HAMT (Chen et al. 2021).

In the first example, both VLN-HAMT and our model made an erroneous action in the 3-rd step. Then, VLN-HAMT did not realize the error and still tried to accomplish the "take a left...take a right..." command in the in-

[4]https://ai.google.com/research/rxr/competition



(a) Ground Truth (b) VLN-HAMT (c) Ours

Figure 6: Qualitative examples comparing VLN-HAMT (Chen et al. 2021) with our agent.

struction, which caused it to totally deviate from the ground truth path. Differently, our agent successfully realized that at the 3-rd position, it was hard to make the left turn, meaning that the last action could be erroneous. So based on the tree-based action space, it chose to backtrack to the 2-nd position and re-choose the correct action this time. Another error occurred later at the 6-th step, where the agent again detected and fixed it. The agent eventually identified the exact ground truth path through these two intermediate error-and-corrections.

The second example showed a more challenging scenario where both VLN-HAMT and our agent made two successive erroneous actions (the 3-rd and 4-th action). VLN-HAMT failed to recover from the error and continued to step forward. However, our agent conducted proper error-correction again and backtracked to the correct path, showing the effectiveness of our proposed designs.

Conclusion

In this work, we proposed a novel tree-structured encoding strategy to replace the previous sequential strategy. Based on it, we designed a Tree-Transformer module to encode the tree structure and further proposed to extend the one-step action space to a tree-based action space, and a new Tree-nDTW reward for better training the agent. Our agent shows surpassing results on the benchmark datasets. We also conducted extensive ablation studies and analyses to further demonstrate the effectiveness of our designs.

Acknowledgments

The research is supported by Science and Technology Innovation 2030 - New Generation Artificial Intelligence (2020AAA0104401), Beijing Natural Science Foundation (Z190001), and Peng Cheng Laboratory Key Research Project No.PCL2021A07.

References

Anderson, P.; Chang, A. X.; Chaplot, D. S.; Dosovitskiy, A.; Gupta, S.; Koltun, V.; Kosecka, J.; Malik, J.; Mottaghi, R.; Savva, M.; and Zamir, A. R. 2018a. On Evaluation of Embodied Navigation Agents. *CoRR*, abs/1807.06757.

Anderson, P.; Shrivastava, A.; Parikh, D.; Batra, D.; and Lee, S. 2019. Chasing Ghosts: Instruction Following as Bayesian State Tracking. In *NeurIPS*, 369–379.

Anderson, P.; Wu, Q.; Teney, D.; Bruce, J.; Johnson, M.; Sünderhauf, N.; Reid, I. D.; Gould, S.; and van den Hengel, A. 2018b. Vision-and-Language Navigation: Interpreting Visually-Grounded Navigation Instructions in Real Environments. In *CVPR*, 3674–3683.

Chang, A. X.; Dai, A.; Funkhouser, T. A.; Halber, M.; Nießner, M.; Savva, M.; Song, S.; Zeng, A.; and Zhang, Y. 2017. Matterport3D: Learning from RGB-D Data in Indoor Environments. In *3DV*, 667–676.

Chen, S.; Guhur, P.; Schmid, C.; and Laptev, I. 2021. History Aware Multimodal Transformer for Vision-and-Language Navigation. *CoRR*, abs/2110.13309.

Chen, S.; Guhur, P.; Tapaswi, M.; Schmid, C.; and Laptev, I. 2022. Think Global, Act Local: Dual-scale Graph Transformer for Vision-and-Language Navigation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, 16516–16526.

Chen, Y.; Li, L.; Yu, L.; Kholy, A. E.; Ahmed, F.; Gan, Z.; Cheng, Y.; and Liu, J. 2020. UNITER: UNiversal Image-TExt Representation Learning. In *ECCV*, 104–120.

Deng, Z.; Narasimhan, K.; and Russakovsky, O. 2020. Evolving Graphical Planner: Contextual Global Planning for Vision-and-Language Navigation. In *NeurIPS*.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, 4171–4186.

Dou, Z.; and Peng, N. 2022. FOAM: A Follower-aware Speaker Model For Vision-and-Language Navigation. In *NAACL*, 4332–4340.

Fried, D.; Hu, R.; Cirik, V.; Rohrbach, A.; Andreas, J.; Morency, L.; Berg-Kirkpatrick, T.; Saenko, K.; Klein, D.; and Darrell, T. 2018. Speaker-Follower Models for Vision-and-Language Navigation. In *NeurIPS*, 3318–3329.

Fu, T.; Wang, X. E.; Peterson, M. F.; Grafton, S. T.; Eckstein, M. P.; and Wang, W. Y. 2020. Counterfactual Vision-and-Language Navigation via Adversarial Path Sampler. In *ECCV*, 71–86.

Guhur, P.-L.; Tapaswi, M.; Chen, S.; Laptev, I.; and Schmid, C. 2021. Airbert: In-Domain Pretraining for Vision-and-Language Navigation. In *ICCV*, 1634–1643.

Hao, W.; Li, C.; Li, X.; Carin, L.; and Gao, J. 2020. Towards Learning a Generic Agent for Vision-and-Language Navigation via Pre-Training. In *CVPR*, 13134–13143.

Harer, J.; Reale, C. P.; and Chin, P. 2019. Tree-Transformer: A Transformer-Based Method for Correction of Tree-Structured Data. *CoRR*, abs/1908.00449.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Comput.*, 9(8): 1735–1780.

Hong, Y.; Opazo, C. R.; Qi, Y.; Wu, Q.; and Gould, S. 2020. Language and Visual Entity Relationship Graph for Agent Navigation. In *NeurIPS*.

Hong, Y.; Wu, Q.; Qi, Y.; Opazo, C. R.; and Gould, S. 2021. VLN BERT: A Recurrent Vision-and-Language BERT for Navigation. In *CVPR*, 1643–1653.

Huang, H.; Jain, V.; Mehta, H.; Ku, A.; Magalhães, G.; Baldridge, J.; and Ie, E. 2019. Transferable Representation Learning in Vision-and-Language Navigation. In *ICCV*, 7403–7412.

Ilharco, G.; Jain, V.; Ku, A.; Ie, E.; and Baldridge, J. 2019. General Evaluation for Instruction Conditioned Navigation using Dynamic Time Warping. In *NeurIPS Workshop*.

Ku, A.; Anderson, P.; Patel, R.; Ie, E.; and Baldridge, J. 2020. Room-Across-Room: Multilingual Vision-and-Language Navigation with Dense Spatiotemporal Grounding. In *EMNLP*, 4392–4412.

Li, G.; Duan, N.; Fang, Y.; Gong, M.; and Jiang, D. 2020a. Unicoder-VL: A Universal Encoder for Vision and Language by Cross-Modal Pre-Training. In *AAAI*, 11336–11344.

Li, J.; Tan, H.; and Bansal, M. 2022. Envedit: Environment Editing for Vision-and-Language Navigation. In *CVPR*, 15386–15396.

Li, L. H.; Yatskar, M.; Yin, D.; Hsieh, C.; and Chang, K. 2019a. VisualBERT: A Simple and Performant Baseline for Vision and Language. *CoRR*, abs/1908.03557.

Li, X.; Li, C.; Xia, Q.; Bisk, Y.; Celikyilmaz, A.; Gao, J.; Smith, N. A.; and Choi, Y. 2019b. Robust Navigation with Language Pretraining and Stochastic Sampling. In *EMNLP-IJCNLP*, 1494–1499.

Li, X.; Yin, X.; Li, C.; Zhang, P.; Hu, X.; Zhang, L.; Wang, L.; Hu, H.; Dong, L.; Wei, F.; Choi, Y.; and Gao, J. 2020b. Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks. In *ECCV*, 121–137.

Liang, X.; Zhu, F.; Zhu, Y.; Lin, B.; Wang, B.; and Liang, X. 2022. Contrastive Instruction-Trajectory Learning for Vision-Language Navigation. In *AAAI*, 1592–1600.

Lu, J.; Batra, D.; Parikh, D.; and Lee, S. 2019. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. In *NeurIPS*, 13–23.

Lu, J.; Goswami, V.; Rohrbach, M.; Parikh, D.; and Lee, S. 2020. 12-in-1: Multi-Task Vision and Language Representation Learning. In *CVPR*, 10434–10443.

Ma, C.; Lu, J.; Wu, Z.; AlRegib, G.; Kira, Z.; Socher, R.; and Xiong, C. 2019a. Self-Monitoring Navigation Agent via Auxiliary Progress Estimation. In *ICLR*.

Ma, C.; Wu, Z.; AlRegib, G.; Xiong, C.; and Kira, Z. 2019b. The Regretful Agent: Heuristic-Aided Navigation Through Progress Estimation. In *CVPR*, 6732–6740.

Majumdar, A.; Shrivastava, A.; Lee, S.; Anderson, P.; Parikh, D.; and Batra, D. 2020. Improving Vision-and-Language Navigation with Image-Text Pairs from the Web. In *ECCV*, 259–274.

Parvaneh, A.; Abbasnejad, E.; Teney, D.; Shi, Q.; and van den Hengel, A. 2020. Counterfactual Vision-and-Language Navigation: Unravelling the Unseen. In *NeurIPS*.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global Vectors for Word Representation. In *EMNLP*, 1532–1543.

Qi, Y.; Pan, Z.; Hong, Y.; Yang, M.-H.; van den Hengel, A.; and Wu, Q. 2021. The Road To Know-Where: An Object-and-Room Informed Sequential BERT for Indoor Vision-Language Navigation. In *ICCV*, 1655–1664.

Qi, Y.; Pan, Z.; Zhang, S.; van den Hengel, A.; and Wu, Q. 2020. Object-and-Action Aware Model for Visual Language Navigation. In *ECCV*, 303–317.

Qiao, Y.; Qi, Y.; Hong, Y.; Yu, Z.; Wang, P.; and Wu, Q. 2022. HOP: History-and-Order Aware Pre-training for Vision-and-Language Navigation. *CoRR*, abs/2203.11591.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving language understanding by generative pre-training.

Su, W.; Zhu, X.; Cao, Y.; Li, B.; Lu, L.; Wei, F.; and Dai, J. 2020. VL-BERT: Pre-training of Generic Visual-Linguistic Representations. In *ICLR*.

Sun, C.; Myers, A.; Vondrick, C.; Murphy, K.; and Schmid, C. 2019. VideoBERT: A Joint Model for Video and Language Representation Learning. In *ICCV*, 7463–7472.

Sun, Z.; Zhu, Q.; Xiong, Y.; Sun, Y.; Mou, L.; and Zhang, L. 2020. TreeGen: A Tree-Based Transformer Architecture for Code Generation. In *AAAI*, 8984–8991.

Tan, H.; and Bansal, M. 2019. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *EMNLP-IJCNLP*, 5099–5110.

Tan, H.; Yu, L.; and Bansal, M. 2019. Learning to Navigate Unseen Environments: Back Translation with Environmental Dropout. In *NAACL-HLT*, 2610–2621.

Thrun, S.; Burgard, W.; and Fox, D. 2005. *Probabilistic robotics*. Intelligent robotics and autonomous agents. MIT Press.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NeurIPS*, 5998–6008.

Wang, H.; Liang, W.; Shen, J.; Gool, L. V.; and Wang, W. 2022. Counterfactual Cycle-Consistent Learning for Instruction Following and Generation in Vision-Language Navigation. In *CVPR*, 15450–15460.

Wang, H.; Wang, W.; Liang, W.; Xiong, C.; and Shen, J. 2021. Structured Scene Memory for Vision-Language Navigation. *CoRR*, abs/2103.03454.

Wang, H.; Wang, W.; Shu, T.; Liang, W.; and Shen, J. 2020a. Active Visual Information Gathering for Vision-Language Navigation. In *ECCV*, 307–322.

Wang, H.; Wu, Q.; and Shen, C. 2020. Soft Expert Reward Learning for Vision-and-Language Navigation. In *ECCV*, 126–141.

Wang, X.; Huang, Q.; Celikyilmaz, A.; Gao, J.; Shen, D.; Wang, Y.; Wang, W. Y.; and Zhang, L. 2019. Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation. In *CVPR*, 6629–6638.

Wang, X.; Xiong, W.; Wang, H.; and Wang, W. Y. 2018. Look Before You Leap: Bridging Model-Free and Model-Based Reinforcement Learning for Planned-Ahead Vision-and-Language Navigation. In *ECCV*, 38–55.

Wang, X. E.; Jain, V.; Ie, E.; Wang, W. Y.; Kozareva, Z.; and Ravi, S. 2020b. Environment-Agnostic Multitask Learning for Natural Language Grounded Navigation. In *ECCV*, 413–430.

Wang, Y.; Lee, H.; and Chen, Y. 2019. Tree Transformer: Integrating Tree Structures into Self-Attention. In *EMNLP-IJCNLP*, 1061–1070.

Zhang, J.; Wei, Z.; Fan, J.; and Peng, J. 2021. Curriculum Learning for Vision-and-Language Navigation. *CoRR*, abs/2111.07228.

Zhu, F.; Zhu, Y.; Chang, X.; and Liang, X. 2020a. Vision-Language Navigation With Self-Supervised Auxiliary Reasoning Tasks. In *CVPR*, 10009–10019.

Zhu, W.; Hu, H.; Chen, J.; Deng, Z.; Jain, V.; Ie, E.; and Sha, F. 2020b. BabyWalk: Going Farther in Vision-and-Language Navigation by Taking Baby Steps. In *ACL*, 2539–2556.