Explore the Potential Performance of Vision-and-Language Navigation Model: a Snapshot Ensemble Method

Anonymous ACL submission

Abstract

Given an instruction in a natural language, the vision-and-language navigation (VLN) task requires a navigation model to match the instruction to its visual surroundings and then move to the correct destination. It has been difficult to build VLN models that can generalize as well as humans. In this paper, we provide a new perspective that accommodates the potential variety of interpretations of verbal instructions. We discovered that snapshots of a VLN model, i.e., model versions based on parameters saved at various intervals during its training, behave significantly differently even when their navigation success rates are almost the same. We thus propose a snapshot-based ensemble solution that leverages predictions provided by multiple snapshots. Our approach is effective and generalizable, and can be applied to ensemble snapshots from different models. Constructed on the mixed snapshots of the existing state-of-the-art (SOTA) RecBERT *and* HAMT models, our proposed ensemble achieves new SOTA performance in the R2R Dataset Challenge in the single-run setting [1].

1 Introduction

With a set of movement instructions provided at the beginning of an agent's navigation task, a Vision-and-Language Navigation (VLN) model guides the agent through an environment that is revealed through visual input one step at a time. Building an effective VLN model is difficult because it needs to understand and coordinate both types of information, vision and language inputs. Recent advancements in computer vision and natural language processing and the advent of better vision-and-language models (Sundermeyer et al. (2012); Vaswani et al. (2017); Lu et al. (2019); Tan and Bansal (2019)) along with the effort to prepare large scale realistic datasets (Chang et al., 2017) have enabled rapid development of VLN systems. Benchmarking VLN models using the R2R dataset (Anderson et al., 2018) that is based on real photos of indoor environments, has been popular, due to its simple-form task, which at the same time requires a complex understanding of both images and text (see Fig. 1). Various studies have discussed how to improve benchmark performance by adjusting model structure (Anderson et al., 2018; Majumdar et al., 2020; Wang et al., 2020; Hong et al., 2021) or adding more complicated mechanisms to the models (Ma et al., 2019b; Zhu et al., 2020; Chen et al., 2021b). Previous studies have also made efforts to prevent overfitting to training data (Daniel Fried et al., 2018; Liu et al., 2021; Li et al., 2019; Hao et al., 2020).

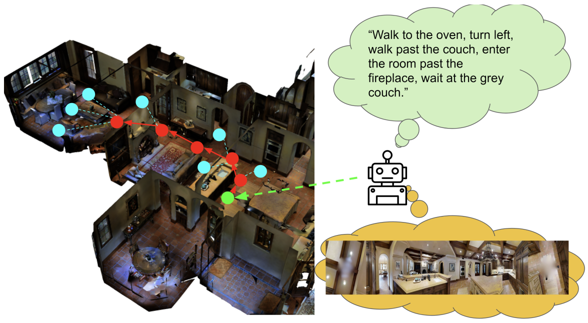


Figure 1: VLN task on R2R data: An agent receives textual navigation instructions at a start node (green cloud) and surrounding views (beige cloud). Controlled by a VLN model, it decides where to go next (correct nodes shown in red, other navigable positions in cyan).

In this paper, we offer a new VLN solution that focuses on the by-products of the model training process: snapshots. Snapshots are versions of a model that are defined by the saved parameters of the model at various intervals during its training. Although all snapshots have the same goal as the model, their trained parameters are different due to the ongoing optimization process. We discovered

[1] The leaderboard can be found at `https://eval.ai/web/challenges/challenge-page/97/leaderboard/270`, and our result is named "SE-Mixed (HAMT+RecBERT) (Single-run)."

that some of the best snapshots at various intervals saved during training shared *similar* navigation success rates while making *significantly diverse* errors. Based on this observation, we constructed our VLN system with an ensemble of snapshots instead of just one. Our experiments revealed that such an ensemble can take advantage of its members and thus exploit the potential variety of interpretations of verbal instructions and their matches to the visual surroundings. As a result, the ensemble significantly improves the navigation performance. We also found that ensembles of snapshots can be further optimized by adding a meta-learner to decide which snapshots should be included in the ensemble. In our case, we set up a beam-search mechanism to do so.

To produce even more variant candidate snapshots to construct the ensemble, we built an ensemble from snapshots of more than one VLN base model. Our experimental results show that snapshots from the different models are supplementary to each other and thus lead to an even better result than snapshot ensembles from only one model.

To conclude, our contributions are as follows:

- We discovered that the best snapshots of a model interpret verbal and visual input differently while having similar navigation success rates. We thus propose a snapshot ensemble method to take advantage of the different snapshots.
- Since not all of the many potential snapshots are beneficial to the ensemble, we proposed a beam-search-based meta-learner that decides the best combination of snapshots to be included in the ensemble in an efficient manner.
- By combining the snapshots from existing VLN models: Recurrent-VLN-BERT (RecBERT) and History Aware Multimodal Transformer (HAMT), our ensemble achieves a new SOTA performance on the R2R challenge leaderboard in the single-run setting.
- Additional experiments with three model architectures and two datasets with different levels of task difficulty show the efficacy and generality of our snapshot ensemble method.

We suggest that our proposed snapshot ensemble process could be applied to other tasks that use natural language, for example, to "navigate" digital domains such as websites (Pasupat et al., 2018) and mobile apps (Li et al., 2020b) or for addressing visual goal-step inference task (Yang et al., 2021).

2 Related Works

Vision-and-language Navigation task and datasets. Teaching a robot to complete instructions is a long-existing goal in the AI community (Winograd, 1971). Different from GPS-based navigation, a VLN system accepts instructions in natural language and matches them to visual inputs from its surrounding environments. Most VLN datasets in the past consist of synthesized 3D scenes (Kolve et al., 2017; Brodeur et al., 2017; Wu et al., 2018; Yan et al., 2018; Song et al., 2017). Recently, the emergence of datasets based on real 3D scenes allows VLN systems to be developed and tested in realistic environments. Specifically, 3D views from Google Street View and Matterport3D datasets (Chang et al., 2017) allow people to build simulators that generate navigation data from photos taken in real life. Different from the previous 3D-synthesized datasets, the R2R dataset (Anderson et al., 2018) that we use consists of navigation task in real indoor environments. Concretely, the R2R dataset provides ∼15,000 instructions and ∼5,000 navigation paths in 90 indoor scenes. Since its publication, researchers have proposed variants of the R2R dataset to address some of its shortcomings (Ku et al., 2020; Jain et al., 2019; Hong et al., 2020b; Krantz et al., 2020). However, the community still considers the R2R dataset to be fundamental in benchmarking indoor VLN systems.

VLN systems using the R2R dataset. To improve navigation performance of the R2R baseline system (Anderson et al., 2018), various models and techniques have been proposed, including using LSTM (Daniel Fried et al., 2018) and soft-attention (Tan et al., 2019). Previous work closest so ours is by Hu et al. (2019), who proposed a mixture of VLN models, each trained with different inputs. Majumdar et al. (2020) proposed a VLN system based on a pre-trained vision and language model VilBERT (Lu et al., 2019). Recently, Chen et al. (2021a); Wang et al. (2021); Hong et al. (2020a) proposed VLN systems based on graph models. Liu et al. (2021) provided data augmentation by splitting and mixing scenes. Ma et al. (2019b,a) introduced regularization loss and back-tracking. Tan et al. (2019) improved the dropout mechanism in their VLN model. Li et al. (2019); Hao et al. (2020) improved the model's initial states by pre-training it on large-scale datasets.

A significant improvement in SOTA performance was achieved by the RecBERT model (Hong et al., 2021), which utilizes the *CLS* token, a special token added in front of every input sequence in BERT-like models (Jacob Devlin et al., 2019), as a recurrent state. We adopted RecBERT as the basic model to illustrate our snapshot ensemble solution due to RecBERT's high performance and easy-to-reproduce code structure.[2] Another high performing model, HAMT (Chen et al., 2021b), uses pretraining based on proxy tasks such as masked word prediction and instruction-trajectory matching and allows an agent's previous actions to be involved in the prediction of the current action. We tested ensembles of HAMT snapshots and also combined it with RecBERT in a mixed-model ensemble.

Ensemble Models. An ensemble of models expands the solution space and has a better chance to avoid local minima (Hansen and Salamon, 1990). It can be created in different ways. Most relevant to our work is the idea of bagging (Breiman, 1996, 2001) which trains the same model with different input data, and stacking (Wolpert, 1992), which focuses on building a meta-learner by optimizing the predictions given by different models in the ensemble.

Our work is inspired by the idea of a "snapshot ensemble" by Huang et al. (2017), which is constructed from a set of snapshots collected at local minima. Zhang et al. (2020) further developed the idea of a snapshot ensemble for classification with boosting and stacking. Different from previous works, we collect snapshots based on training intervals and performance. We apply beam-search as the meta-learner that optimizes the choices of snapshots to be included in the ensemble.

3 Method

3.1 Vision-and-language Navigation in R2R

Navigation in R2R consists of three parts: instruction I, scene S, and path P. The instruction I is a sequence of L words in the vocabulary W: $I = \{w_1, w_2, ..., w_L \,|w_i \subset W, 1 \leq i \leq L\}$. The instructions are all manually labeled with a WebGL interface that displays 3D scenes constructed from the Matterport3D dataset (Chang et al., 2017). The instruction I describes the navigation path P based on the surrounding views along the path, without aligning specific words to a particular viewpoint, making the task even more challenging. The scene $S = \{V, E\}$ is a connected graph of viewpoints V and the edges E that connect viewpoints. The path P is a sequence of viewpoints in S i.e., $P = \{v_1, v_2, ..., v_n | v \in V\}$ from start v_1 to destination v_n. At any time during navigation, an agent is placed in a certain viewpoint $v_i \in V$. For each viewpoint v_i, there is a corresponding panoramic view O_i to describe the visual surroundings of v_i. For the RecBERT model, views in O_i are converted to image features by a pre-trained ResNet-152 model.

To complete a *single-run* R2R navigation task, a VLN model controls the agent's movements in S from v_1 to v_n in one pass with as few steps as possible. The model works as a policy function π with the instruction I and the panoramic view O_i of viewpoint v_i as inputs. At each time step t, the policy function predicts an action $a_t \leftarrow \pi(I, O_i)$ that moves the agent to a navigable viewpoint or stop the navigation. If the last viewpoint v_{end} is within a certain distance (3 m) to the endpoint v_n of the ground-truth path P, the navigation is considered to be successful, otherwise it is considered as failed. The performance of a VLN model is mainly based on how many successful navigations it recommends during evaluation, namely the "success rate" (additional metrics in Section 5.1).

3.2 Snapshots of the Same Model

When designing a supervised learning model, we usually choose the most accurate snapshot found in the validation process to represent the trained model and discard the other snapshots. We discovered, however, such discarded snapshots are valuable in improving the task performance of the model. In this section, we adopt the RecBERT model as an example to illustrate how we discover the uses of snapshots saved during training.[3] A more detailed explanation of the RecBERT model is given in Appendix A.

We first trained RecBERT and measured its validation success rates on navigations in environments that it had never seen before, called "val_unseen split." We noticed that the success rates fluctuate drastically over time (Fig. 2). We also observed that both imitation and reinforcement learning losses drop consistently with time (and equally, success rates on *seen* environments increase consistently with time). These interesting discoveries led us to

[2]https://github.com/YicongHong/Recurrent-VLN-BERT

[3]Here, we call RecBERT initialized by PREVALENT (Hao et al., 2020) simply "RecBERT," and the model initialized by OSCAR (Li et al., 2020a) "OSCAR-initialized RecBERT."

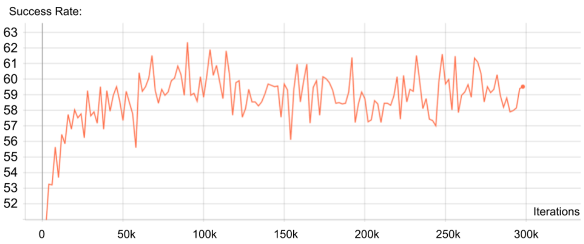


Figure 2: The curve of *validation* success rate over time during training. We can observe a drastic fluctuation throughout the training.

Snapshot Period	Success Rate in val_unseen Split
90K - 120K	62.32%
240K - 270K	61.60%
210K - 240K	61.56%
60K - 90K	61.52%
180K - 210K	61.30%

Table 1: Navigation success rates for the top-5 snapshots of RecBERT in 10 periods of a 300,000-iteration training cycle.

further investigate whether snapshots that perform similarly in terms of success rates might behave differently with respect to the errors that they make.

We set up an experiment designed as follows: we trained the RecBERT model for 300,000 iterations and saved the best snapshot in the validation split for every 30,000 iterations (Table 1). We chose the best two snapshots (62.32% and 61.60% success rates) and counted the navigations for which only one of the snapshots failed, both of the snapshots failed, or none failed. Our results show that 563 navigations ended with different results between the best and the second-best snapshots, approximately 24% of the validation data. In comparison, the difference in their success rates is only 0.72%. The massive difference between 24% and 0.72% suggests that different navigation recommendations occur even though success rates are almost equal.

We also discovered that different snapshots may pay attention to different words in the instruction at the same time step even though their predicted paths may be identical. To study this, we added an attention regularization loss on RecBERT during training (details in Appendix B) that encourages the model to pay attention to the sub-instruction that corresponds to the ground-truth path viewpoint at each step (the ground-truth sub-instruction information is provided by the "Fine-grained R2R" dataset (Hong et al., 2020b)). We found that the attention regularization does not bring significant increase or decrease of performance to the model, but the attention scores enable us to see which words the model focuses on in each step. The different

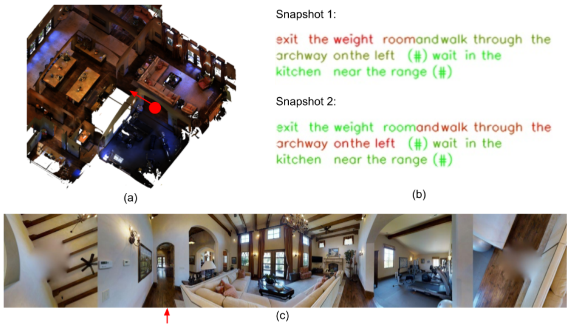


Figure 3: (a) Scene with current position of agent (red). (b) Attention scores for words by two snapshot models from different training periods (high attention in red, low in green). (c) Panoramic view of the agent at the current position. Interestingly, both snapshots make the same movement recommendation (red arrow in (a) and (c)), although the attention scores visualized in (b) suggest that the two snapshots focused on different words.

distributions of high attention words between the two snapshots of the same model suggest these snapshots look at different words when facing an identical instruction and surroundings (Fig. 3). We next describe how we can leverage the behaviors of multiple snapshots in an ensemble and thus create a better agent.

4 Proposed Snapshot Ensemble Method

Our proposed method consists of three algorithms, a snapshot builder (Algorithm 1), a procedure to use the ensemble to decide on the next navigation step (Algorithm 2), and a method to select an ensemble (Algorithm 3). The snapshot builder ensures that M snapshots are evenly selected during model training on the validation data. Algorithm 2 computes the textual and visual embeddings x_i, y_i per snapshot s_i of the basic model (e.g., RecBERT or HAMT) and the action recommendation $s_i(x_i, y_i)$ at a given step of the navigation process, i.e., for a given viewpoint v. The action recommendation is a vector of scores, where each entry corresponds to a particular action available at viewpoint v. Algorithm 2 then computes a cumulative score $p(a_j)$ for each action a_j by adding the recommendations of all ensemble snapshots for that action. Finally, Algorithm 2 returns the action a_{ensemble} with the highest cumulative score as the action recommended by the ensemble.

Running a single RecBERT model at inference time costs a certain amount of time and memory that scales up quickly when the number of snapshots included in the ensemble increases. Furthermore, some resources may not be used effectively

Algorithm 1 Building Snapshots for the Ensemble

procedure SNAPSHOT BUILDER(Model,Validation-Split)
  Divide a training process of N epochs evenly into M periods $\{m_1, m_2, ..., m_M\}$, assuming N is divisible by M.
  while Training the model for N epochs **do**
    for i from 1 to M **do**
      During each period m_i, save the snapshot s_i with the highest success rate
  return $\{s_1, \ldots s_M\}$,

Algorithm 2 Navigation with Ensemble

procedure NAVIGATION NEXT STEP(Viewpoint v, instruction x, snapshots $s_1, \ldots, s_M$)
  if viewpoint $v = v_{end}$ **then** Exit
  for each s_i **do**
    Compute textual feature x_i
    Compute visual feature y_i at v
  for each action a_j available at v **do**
    Compute score $p(a_j) = \sum_1^M s_i(x_i, y_i)$
  $a_{\text{ensemble}} = \arg\max\{\forall a_j \mid p(a_j)\}$
  return a_{ensemble}

since not all snapshots are contributing equivalently to the improvement of the ensemble performance. We therefore needed to find an *efficient and effective* method to build an ensemble. We propose a beam search procedure (Algorithm 3) as a "meta-learner" to select only a subset of the saved snapshots to be included in the ensemble. There are several benefits of applying beam search as a meta learner: It does not need training. The search only takes time at evaluation, which is much less costly than training a meta-learner. Also, to set up an ensemble of size k, with beam size l, the approximate number of evaluations needed for our beam search strategy is $O(Mlk)$ when $M \gg k$, which is much smaller than the cost of an exhaustive search $O(\min(M^k, M^{(M-k)})$.

An alternative way to set up an ensemble without searching is to choose the top-k saved snapshots. Our investigation shows that an ensemble of top-3 snapshots only achieves 63.5% success rate on val_unseen split, while the best ensemble of size 3 found by the beam search process achieved 65.4%, almost 2 pp better. We suggest that our proposed beam search process has a good balance between efficiency and performance.

5 Experiments

We ran the following experiments to evaluate the performances of snapshot ensembles in different models and datasets:

(1) We evaluated the performance of snapshot

Algorithm 3 Select Snapshots to Build an Ensemble

procedure META-LEARNING ENSEMBLE SELECTOR(Model Snapshots $s_1, \ldots, s_M$)
  Let $S_{candidate} = \{s_1, \ldots, s_M\}$.
  Let $B \leftarrow [\,]$ ▷ B keeps track of the top-l ensembles.
  Add $S_1, S_2, ..., S_l = \{\}$ to B. ▷ l is the beam size
  Set $k \leftarrow 1$.
  while $k \leq K$ **do** ▷ K is max size of ensemble.
    for $s_i \in S_{candidate}$ **do**
      for $S_j \in B$ **do**
        if s_i not in S_j **then**
          Evaluate $\{s_i\} + S_j$
    $B \leftarrow$ the top-l ensembles from all $\{s_i\} + S_j$.
    $k \leftarrow k + 1$
  return Best ensemble ever saved in B. ▷ It is not necessarily in the most recently updated B.

ensembles on the R2R dataset, including ensembles built from RecBERT model snapshots, HAMT model snapshots, and from both. A detailed explanation of the HAMT model is given in Appendix C.

(2) We created snapshot ensembles with other VLN models, namely the OSCAR-initialized RecBERT (Li et al., 2020a) and Env-Drop (Tan et al., 2019). We compared their ensemble performances on R2R against their best single snapshot.

(3) We evaluated the performance of the RecBERT snapshot ensemble on the R4R dataset, which is a larger VLN dataset than R2R and contains more complicated navigation paths.

5.1 Dataset Setting and Evaluation Metrics

We used the R2R train split as training data, val_unseen split as validation data, and test split to evaluate the ensemble. For the R4R dataset, we also used the train split as the training data. As there is no test split in the R4R dataset, we divided its val_unseen split into two halves that do not share scenes. We constructed the snapshot ensemble on one half and evaluated it on the other half.

We adopted four metrics for evaluation: Success Rate (SR), Trajectory Length (TL), Navigation-Error (NE), and Success weighted by Path Length (SPL). SR is the ratio of successful navigation numbers to all navigations (higher is better). TL is the average length of the model's navigation path (lower is better). NE is the average distance between the last viewpoint in the predicted path and the ground truth destination viewpoint (lower is better); SPL is the path-length weighted success rate compared to SR (higher is better).

5.2 Training Setting and Hard/Software Setup

We trained the RecBERT and the OSCAR-initialized RecBERT with a default 300,000 iter-

Model		R2R val_unseen				R2R test			
		TL (↓)	NE (↓)	SR (↑)	SPL (↑)	TL (↓)	NE (↓)	SR (↑)	SPL (↑)
Random	Anderson et al. (2018)	9.77	9.23	16	-	9.89	9.79	13	12
Human	Anderson et al. (2018)	-	-	-	-	11.85	1.61	86	76
Seq2Seq-SF	Anderson et al. (2018)	**8.39**	7.81	22	-	**8.13**	7.85	20	18
Speaker-Follower	Daniel Fried et al. (2018)	-	6.62	35	-	14.82	6.62	35	28
PRESS	Li et al. (2019)	10.36	5.28	49	45	10.77	5.49	49	45
EnvDrop	Tan et al. (2019)	10.7	5.22	52	48	11.66	5.23	51	47
AuxRN	Zhu et al. (2020)	-	5.28	55	50	-	5.15	55	51
PREVALENT	Hao et al. (2020)	10.19	4.71	58	53	10.51	5.3	54	51
RelGraph	Hong et al. (2020a)	9.99	4.73	57	53	10.29	4.75	55	52
RecBERT	Hong et al. (2021)	12.01	3.93	63	57	12.35	4.09	63	57
OSCAR-init. RecBERT	Hong et al. (2021)	11.86	4.29	59	53	12.34	4.59	57	53
RecBERT + REM	Liu et al. (2021)	12.44	3.89	63.6	57.9	13.11	3.87	65.2	59.1
HAMT	Chen et al. (2021b)	11.46	2.29	66	61	12.27	3.93	65	60
Ours:									
EnvDrop Snapshot Ensemble		11.74	4.9	53.34	49.49	11.9	4.98	53.58	50.01
RecBERT Snapshot Ensemble		11.79	3.75	65.55	59.2	12.41	4	64.22	58.96
OSCAR-init. RecBERT Snapshot Ensemble		11.22	4.21	59.73	54.76	11.74	4.36	59.72	55.35
HAMT Snapshot Ensemble		11.67	3.44	67.82	62.27	12.47	3.77	66.45	61.07
RecBERT + HAMT Mixed Snapshot Ensemble		10.96	**3.20**	**70.58**	**65.24**	11.79	**3.52**	**69.82**	**64.66**

Table 2: Evaluation results (best performance bolded). Our mixed snapshot ensemble achieved the new SOTA performance in NE, SR, and SPL.

ations. We ran an ablation study to decide M = 10, k = 4 and l = 3 for constructing the ensemble (we fixed l to be 3 and fine-tuned M and k, detailed in Appendix D). When mixing the RecBERT and HAMT models, the candidate number becomes $2M$ accordingly. In R4R, we set $k = 3$ to shorten the evaluation time. For other parameters, we used the default given by the authors.[4] As for the training of the HAMT model, the model is initialized from the end-to-end, with the proxy-task-finetuned states provided by their source code. [5]

We ran the training code under Ubuntu 20.04.1 LTS operating system, GeForce RTX 3090 Graphics Card with 24GB memory. It takes around 10,000 MB of graphics card memory to evaluate an ensemble of 4 snapshots with batch size 8 inputs. The code was developed in Pytorch 1.7.1, and CUDA 11.2. The training takes approximately 30–40 hours. The beam search evaluation was done in 3–5 hours for R2R and twice that time for R4R.

6 Results

Results on R2R. We evaluated the snapshot ensemble of the RecBERT model, the HAMT models, and a mix of them on the R2R test split (Table 2). All snapshot ensembles show improved performance NE, SR, and SPL metrics over single snapshots. The mixed snapshot ensemble (last row) improved the performance by almost 5 percent points (pp) in SR and SPL, showing that snapshots across models have a good synergy with each other.

In our second set of experiments that evaluated whether other VLN models can be improved with a snapshot ensemble, we found that both ensembles (based on OSCAR-init RecBERT and EnvDropout) consistently gained more than 2 pp increase in SR and SPL compared to the best snapshot of the respective models (Table 2). That suggests the snapshot ensemble is also able to improve the performances of VLN models with different structures.

Results on R4R. The more challenging dataset R4R (Jain et al., 2019) contains more data and more complicated paths of variant lengths. We saw a more than 1 pp increase in SR and SPL after applying the snapshot ensemble (Table 3).

7 Discussion

We now discuss why a snapshot ensemble works well for VLN. We use a RecBERT ensemble of size 3 as an example for investigation.

7.1 Ensemble Balances Snapshot Predictions

Linguistic understanding errors made by one or more snapshots of the ensemble can be corrected by the others because the ensemble predicts actions based on a weighted voting mechanism, whose voters are the snapshot scores ($s_i(x_i, y_i)$ in line 7 of Algorithm 2) as the weights. We give an example in Fig. 4. At the second step of the navigation, two of the snapshots falsely misinterpreted the words

[4] We do not adopt the cyclic learning rate schedule (Ilya Loshchilov and Frank Hutter, 2017) suggested by Huang et al. (2017) that forces the model to generate local minima because we found no significant improvement in a trial.

[5] https://github.com/cshizhe/VLN-HAMT

Model	R4R val_unseen_half				R4R val_unseen_full			
	TL↓	NE↓	SR↑	SPL↑	TL↓	NE↓	SR↑	SPL↑
Speaker-Follower	-	-	-	-	19.9	8.47	23.8	12.2
EnvDrop	-	-	-	-	-	9.18	34.7	21
RecBERT	13.76	7.05	37.29	27.38	13.92	6.55	43.11	32.13
RecBERT Snapshot Ensemble (ours)	15.09	**7.03**	**39**	**28.66**	14.71	**6.44**	**44.55**	**33.45**

Table 3: Results on R4R with half and full splits. The ensemble gains in all metrics over RecBERT.

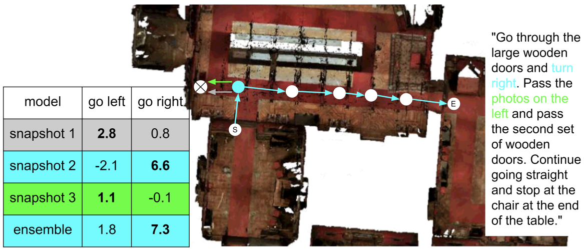


Figure 4: The ensemble makes the correct decision despite linguistic misunderstandings by some snapshots. The correct path from (S) to (E) is recommended by a high-confidence snapshot 2 (cyan) that focuses on "turn right," while snapshots 1 and 3 (green, gray) misinterpret "photos on the left" to mean "turn left."

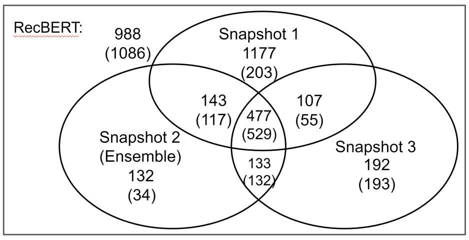


Figure 5: A Venn diagram of the number of failed navigations by RecBERT snapshots. The numbers *not* in any circle are successful navigations by all 3 snapshots. The numbers in parenthesis are the counts when snapshot 2 is replaced by the ensemble, showing that the ensemble share more similar navigations to those of its members than the members' navigations are to each other.

"photos on the left" as a signal for turning left. Due to the weighted voting mechanism, the one snapshot that correctly understood "turn right" in the previous sentence prevents the ensemble from making a mistake. A detailed case study of this correction process is given in Appendix E.

We also observe that the ensemble makes more similar decisions to its snapshots than the snapshots to each other showing its robustness to the differing opinions of its snapshots. To illustrate this observation, we studied its failed navigations compared to the failed navigations of its snapshots. Let s_1, s_2, s_3, s_{ens} represent snapshots 1–3, and the ensemble. Let E be the counts of failed navigations (in val_unseen split). We compute $E_{s_1}, E_{s_2}, E_{s_3}, E_{s_1 \cap s_2}, E_{s_1 \cap s_3}, E_{s_2 \cap s_3}, E_{s_1 \cap s_2 \cap s_3}$, as shown in the Venn diagram in Fig. 5. Then we repeat this process, replacing s_2 with s_{ens}. The ensemble shares more navigations with both snapshots 1 and 3 than snapshot 2 in both successful and failed navigations (i.e., 529 + 1086 shared navigations for the ensemble v.s. 477 + 988 shared navigations for snapshot 2). Meanwhile, the number of navigations that are *only* failed by the ensemble is less than that of snapshot 2 ($34 < 132$). These numbers suggest that the ensemble behaves more similarly to its snapshot members than the replaced snapshot. We repeated this process by replacing snapshots 1 and 3 with the ensemble (one at a time) and also found that the ensemble makes more similar decisions to its snapshots than the snapshots to each other. We also observed this when we used a size-3 mixed snapshot ensemble with RecBERT and HAMT models (see Fig. 9 in Appendix).

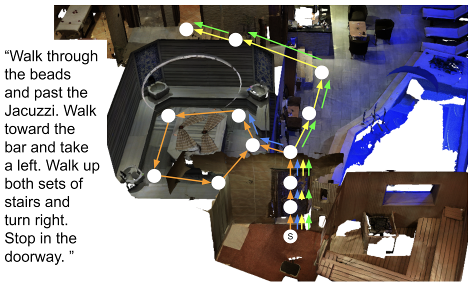


Figure 6: A failed navigation. The ensemble (in orange) is misled by one (in blue) of its three snapshots. The ensemble chose to go left and ignored the correct decisions by the other snapshots (in yellow and green).

However, there are cases where the weighted voting mechanism may lead to an incorrect decision (Fig. 6). When a snapshot makes a wrong decision with high confidence, the prediction may override the recommendations of the rest of the snapshots and lead the ensemble to an incorrect decision. Fortunately, this number of failed ensemble navigations caused by a single snapshot is only about a quarter of all failed navigations and about 10% of the total number of navigations.

To show the advantage of applying an ensemble, we also counted the successful navigations of the ensemble/snapshots in each scene of the dataset (Table 4). We found that different snapshots are

Scene	Ensemble	Snap 1	Snap 2	Snap 3
1	178	165	**169**	159
2	32	**33**	32	29
3	140	131	131	**144**
4	208	189	**199**	185
5	10	**11**	8	9
6	169	161	**170**	152
7	203	198	**200**	196
8	217	205	204	**212**
9	93	80	**89**	84
10	102	**95**	89	89
11	185	177	173	**181**

Table 4: The count of successful navigations for the ensemble and its snapshots (snaps) in each scene on val_unseen split. Best snapshot performances are in bold. We can see that different snapshots are good at different scenes and that the ensemble either outperforms i.e., has more successful navigations than the best snapshot or is comparable to it.

good at different scenes. The ensemble either outperforms its snapshots or is comparable to the best snapshot, suggesting that the ensemble leverages the advantages of snapshots in different scenes to achieve better performance.

7.2 Ensemble Avoids More Long Navigations

Ambiguity always exists in human language. We found that another benefit of an ensemble is that, as long as there is one snapshot that is able to confidently disambiguate, e.g., focus on a keyword not being paid attention to by the others, its prediction can override those almost-tie predictions from other snapshots. For the example in Fig. 7, the instruction "walk straight down kitchen into hallway" can lead to two different paths. If acting individually, two of the three snapshots will recommend an infinite-loop path in the living room (in green and blue). One high-scoring snapshot (in orange) focused more on the word "kitchen" than the phrases "walk straight down" and "into hallway." The ensemble is thus able to recognize the correct path (in red) leading through the kitchen instead of the living room. Generally, we observe that linguistic ambiguity often causes agents to become lost or stuck in infinite loops, and navigation needs to be cut off after a certain number of steps. We use 15 as the default cut-off threshold and call any sequence of recommended actions that is longer than 15 a *Long Navigation* (LN). To quantitatively show how an ensemble prevents more LNs than a single snapshot, we count the LNs for snapshots of the size-3 RecBERT ensemble, and compute the success rates when their navigation is an LN (Table 5). We discovered that an average of 8.13% of the navigations from the snapshots are LNs. The situation is improved in the size-3 RecBERT ensemble, with only 5.5% of its navigations being LNs. Since LN has a high likelihood (> 90%) of failing and the ensemble has significantly fewer LNs than its snapshots (131 vs. up to 246), we consider avoiding more LNs as one of the reasons why the ensemble outperforms single snapshots.

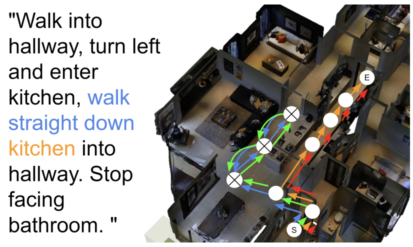


Figure 7: A snapshot ensemble prevents long navigations by disambiguating instructions. The ground truth path (in red) from S to E is recommended by a high-confidence snapshot (orange) that focuses on the word "kitchen." In the ensemble, this snapshot overrides the recommendations of the other two snapshots (green and blue) that focus on "walk straight down" and would lead the agent into an infinite loop (nodes with cross).

	SR	LN Count	LN that fail (%)
Snapshot 1	61.5	172	159 (92.%)
Snapshot 2	62.3	155	141 (91%)
Snapshot 3	61.3	246	223 (91%)
Ensemble	**65.4**	**131**	**123** (94%)

Table 5: Long navigation (LN) count and success ratio (SR). The ensemble is more successful with fewer long navigations than individual snapshots.

8 Conclusion

In this work, we discovered and utilized differences in snapshots of models that make movement recommendations for vision-language navigation. We proposed a snapshot ensemble method that leverages these differences. By combining snapshots of the RecBERT and HAMT models, our method achieves a new SOTA performance on the R2R benchmark dataset. Additional experiments show the generality of our method when applied to other model architectures or data. In future work, we will adapt our snapshot ensemble method to address related navigation tasks that combine vision and language input. We will consider the task of following natural language instructions for navigating digital domains such as websites (Pasupat et al., 2018) or mobile apps (Li et al., 2020b). Snapshot ensembles may also be effective in solving the visual goal-step inference task (Yang et al., 2021).

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.

Leo Breiman. 1996. Bagging Predictors. *Machine Learning*, 24(2):123–140.

Leo Breiman. 2001. Random Forests. *Machine Learning*, 45(1):5–32.

Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville. 2017. Home: A Household Multimodal Environment. *arXiv preprint arXiv:1711.11017*. 6 pp.

Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. 2017. Matterport3D: Learning from RGB-D Data in Indoor Environments. *International Conference on 3D Vision (3DV)*. 25 pp.

Kevin Chen, Junshen K Chen, Jo Chuang, Marynel Vázquez, and Silvio Savarese. 2021a. Topological Planning with Transformers for Vision-and-language Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11276–11286.

Shizhe Chen, Pierre-Louis Guhur, Cordelia Schmid, and Ivan Laptev. 2021b. History Aware Multimodal Transformer for Vision-and-language Navigation. In *35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia*. 14 pp.

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 *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3318–3329.

Lars Kai Hansen and Peter Salamon. 1990. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Weituo Hao, Chunyuan Li, Xiujun Li, Lawrence Carin, and Jianfeng Gao. 2020. Towards Learning a Generic Agent for Vision-and-language Navigation via Pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13137–13146.

Yicong Hong, Cristian Rodriguez, Yuankai Qi, Qi Wu, and Stephen Gould. 2020a. Language and Visual Entity Relationship Graph for Agent Navigation. *Advances in Neural Information Processing Systems*, 33:7685–7696.

Yicong Hong, Cristian Rodriguez, Qi Wu, and Stephen Gould. 2020b. Sub-Instruction Aware Vision-and-language Navigation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3360–3376.

Yicong Hong, Qi Wu, Yuankai Qi, Cristian Rodriguez Opazo, and Stephen Gould. 2021. VLN BERT: A Recurrent Vision-and-Language BERT for Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1643–1653.

Ronghang Hu, Daniel Fried, Anna Rohrbach, Dan Klein, Trevor Darrell, and Kate Saenko. 2019. Are you Looking? Grounding to Multiple Modalities in Vision-and-Language Navigation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6551–6557, Florence, Italy. Association for Computational Linguistics.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. 2017. Snapshot Ensembles: Train 1, Get M for Free. *arXiv preprint arXiv:1704.00109*.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017*. 16 pp.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Vihan Jain, Gabriel Magalhaes, Alexander Ku, Ashish Vaswani, Eugene Ie, and Jason Baldridge. 2019. Stay on the Path: Instruction Fidelity in Vision-and-language Navigation. In *Proc. of ACL*. 11 pp.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. 2017. Ai2-thor: An Interactive 3D Environment for Visual AI. *arXiv preprint arXiv:1712.05474*. 4 pp.

Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. 2020. Beyond the Nav-graph: Vision-and-language Navigation in Continuous Environments. In *European Conference on Computer Vision*, pages 104–120. Springer.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. 2020. Room-Across-Room: Multilingual Vision-and-language Navigation with Dense Spatiotemporal Grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4392–4412, Online. Association for Computational Linguistics.

Xiujun Li, Chunyuan Li, Qiaolin Xia, Yonatan Bisk, Asli Celikyilmaz, Jianfeng Gao, Noah A. Smith, and Yejin Choi. 2019. Robust navigation with language pretraining and stochastic sampling. 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 1494–1499, Hong Kong, China. Association for Computational Linguistics.

Xiujun Li, Xi Yin, Chunyuan Li, Pengchuan Zhang, Xiaowei Hu, Lei Zhang, Lijuan Wang, Houdong Hu, Li Dong, Furu Wei, et al. 2020a. Oscar: Object-semantics Aligned Pre-training for Vision-language Tasks. In *European Conference on Computer Vision*, pages 121–137. Springer.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020b. Mapping Natural Language Instructions to Mobile UI Action Sequences. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8198–8210, Online. Association for Computational Linguistics.

Chong Liu, Fengda Zhu, Xiaojun Chang, Xiaodan Liang, Zongyuan Ge, and Yi-Dong Shen. 2021. Vision-language Navigation with Random Environmental Mixup. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1644–1654.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks. *Advances in Neural Information Processing Systems*, 32:13–23.

Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming Xiong. 2019a. Self-monitoring Navigation Agent via Auxiliary Progress Estimation. *arXiv preprint arXiv:1901.03035*. 18 pp.

Chih-Yao Ma, Zuxuan Wu, Ghassan AlRegib, Caiming Xiong, and Zsolt Kira. 2019b. The Regretful Agent: Heuristic-aided Navigation Through Progress Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6732–6740.

Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. 2020. Improving Vision-and-language Navigation with Image-text Pairs from the Web. In *European Conference on Computer Vision*, pages 259–274. Springer.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR.

Panupong Pasupat, Tian-Shun Jiang, Evan Liu, Kelvin Guu, and Percy Liang. 2018. Mapping Natural Language Commands to Web Elements. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4970–4976, Brussels, Belgium. Association for Computational Linguistics.

Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. 2017. Semantic Scene Completion from a Single Depth Image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 9 pp.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM Neural Networks for Language Modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, pages 194–197.

Hao Tan and Mohit Bansal. 2019. LXMERT: Learning Cross-Modality Encoder Representations from Transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 12 pp.

Hao Tan, Licheng Yu, and Mohit Bansal. 2019. Learning to Navigate Unseen Environments: Back Translation with Environmental Dropout. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2610–2621.

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.

Hanqing Wang, Wenguan Wang, Wei Liang, Caiming Xiong, and Jianbing Shen. 2021. Structured Scene Memory for Vision-language Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8455–8464.

Hanqing Wang, Wenguan Wang, Tianmin Shu, Wei Liang, and Jianbing Shen. 2020. Active Visual Information Gathering for Vision-language Navigation. In *European Conference on Computer Vision*, pages 307–322. Springer.

Terry Winograd. 1971. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Technical report, Massachusetts Institute of Technology, Project MAC.

David H Wolpert. 1992. Stacked Generalization. *Neural Networks*, 5(2):241–259.

Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. 2018. Building Generalizable Agents with a Realistic and Rich 3D Environment. *arXiv preprint arXiv:1801.02209*. 15 pp.

Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. 2018. Chalet: Cornell House Agent Learning Environment. *arXiv preprint arXiv:1801.07357*. 5 pp.

Yue Yang, Artemis Panagopoulou, Qing Lyu, Li Zhang, Mark Yatskar, and Chris Callison Burch. 2021. Visual Goal-Step Inference Using wikiHow. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2167–2179, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Wentao Zhang, Jiawei Jiang, Yingxia Shao, and Bin Cui. 2020. Snapshot Boosting: a Fast Ensemble Framework for Deep Neural Networks. *Science China Information Sciences*, 63(1):1–12.

Fengda Zhu, Yi Zhu, Xiaojun Chang, and Xiaodan Liang. 2020. Vision-language Navigation with Self-supervised Auxiliary Reasoning Tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11 pp.

A RecBERT Model

Recurrent-VLN-BERT (RecBERT) model by Hong et al. (2021) takes as input in each time step textual, visual, and previous state tokens and output action scores using a cross-model self-attention mechanism. A visualization of the RecBERT model structure is given in Figure 8.

When a BERT model converts text inputs into word embeddings for computation, a cls token is added to the beginning of the embedding vector and a sep token is added to its end to indicate the text sequence is over. The cls token will later interact with the words of the instruction, visual features by the attention mechanism in BERT. In RecBERT, the text-and-visual encoded cls token is used to decide what action to take at the current time step. Concretely, an instruction is converted to word embeddings pre-trained by the PREVALENT model (Hao et al., 2020).

Before computing the prediction of actions, the model selects a set of candidate views from O_i. Each candidate view contains a unique navigable viewpoint that leads to the next viewpoint: $O_{cand} = [O_{c1}, O_{c2}, \ldots] \subset O_i$. The O_{cand} will be converted to the ResNet-152 features pre-trained on Place365 dataset with an all-zero vector that represents the 'stop' action: $F^t_{cand} =$ ResNet-152$(O_{cand}) + [O_{stop}]$[6].

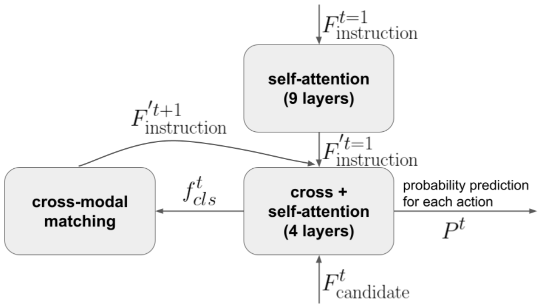


Figure 8: A visualization of the RecBERT model. The instruction feature first passes through a self-attention module and then attends to a candidate feature vector through a cross-self-attention module. The candidate feature then performs self-attention in the same module. After four layers of computation, the last layer outputs the probabilities of each action and sends the cls feature to a cross-modal matching module. The output replaces the cls feature in the instruction vector of the next time step.

After that, the RecBERT model projects the candidate views and the instruction into the same feature space $F_{instruction}, F_{candidate}$. Eventually, we have a vector of instruction features

$$F^{t=1}_{instruction} = [f_{cls}, f_{w_1}, .., f_{w_L}, f_{sep}]$$

and a vector of candidate action features

$$F^{t=1}_{candidate} = [f_{a_1}, ..., f_{a_n}, f_{a_{stop}}]$$

as inputs of the action prediction.

At the first time step, $F^{t=1}_{instruction}$ is sent to a 9-layer self-attended module. Thus the $f^{t=1}_{cls}$ feature is encoded with the information from words of the instruction.

The model then appends $f^{t=1}_{cls}$ to $F^{t=1}_{candidate}$ from $F^{t=1}_{instruction}$. After that, a cross-attention sub-module attends to the remaining elements in $F^{t=1}_{instruction}$, i.e., both $F^{t=1}_{candidate}$ and $f^{t=1}_{cls}$.

Lastly, another sub-module computes the self-attention of the instruction-attended $[F^{t=1}_{candidate}, f^{t=1}_{cls}]$. Such cross and self sub-modules build up the 'cross + self-attention' module in Figure 8. The process repeats for four layers and the attention scores between $f^{t=1}_{cls}$ and each elements in $F^{t=1}_{candidate}$ of the last layer are the prediction scores of each action $p_1, \ldots, p_{stop}$.

[6]In practice, the values for heading and elevation angles of the camera are also concatenated with the image features to encode the relative position of the view in the viewpoint.

Additionally, the f_{cls}^{t} in the output is sent to a cross-modal-matching module. The output of the module is used as f_{cls}^{t+1} in the next time step while other features in $F_{instruction}^{t=1}$ remain unchanged. The 'cross + self-attention' computation will be repeated to compute action predictions for the remaining time steps.

The RecBERT model minimizes two losses, the imitation learning loss and the reinforcement learning loss:

$$\mathcal{L}_{\text{original}} = -\lambda \sum_{t=1}^{T} a_t \log(p_t) - \sum_{t=1}^{T} a_s \log(p_t) A(t), \tag{1}$$

where a_t is the teacher action (one-hot encoded action that gets closest to the destination), p_t is the probability of the taken action, a_s is the action taken, and $A(t)$ is the advantage value at time step t, computed by the A2C algorithm Mnih et al. (2016). To balance the contributions of the imitation and reinforcement learning loss values in the computation of the total loss, hyper-parameter $\lambda = 0.5$ is used.

B Our Inclusion of Attention Regularization in RecBERT

In this section, we describe how we added an attention regularization mechanism to the RecBERT model. The benefit of our approach is that it enables us to monitor which words in the VLN instruction the model pays attention to.

During the computation of the cross-attention that encodes the $F_{candidate}^{t}$ and f_{cls}^{t} with the information from $F_{instruction}^{t}$ the attention scores between the cls token and each word in the instruction are also computed. Hong et al. (2021) observed that the OSCAR-initialized RecBERT model maintains high attention scores on words that correspond to the current navigation step, implying that those words are important to the current decision. Inspired by this observation, we wanted the RecBERT model to have such a feature as well, so that it will be clearer for us to know which words mostly affect the decision of the model.

Concretely, at time step i, for each set of attention scores $X_i = [x_1, ..., x_L]$ from f_{cls}^{i} to each word in the instruction $w_1, .., w_L$, to force such a pattern to be trained, which is defined as follows:

$$\mathcal{L}_{\text{attention}_i} = \text{MSE}(\tanh(X_i), G_i), \tag{2}$$

where "MSE" stands for Mean-Squared-Error and $G_i = [g_{i,1}, ..., g_{i,L}]$ is the "ground truth" values for the normalized attention scores $\tanh(X_i)$. G_i is computed based on the sub-instruction annotation from the Fine-Grained R2R dataset (FGR2R) (Hong et al., 2020b). The FGR2R dataset divides the instructions in the R2R dataset into a set of ordered sub-instructions: $I = [I_{sub_1}, I_{sub_2}, ..., I_{sub_q}]$ where q is the number of sub-instructions the original instruction consists of. Each sub-instruction corresponds to one or a sequence of viewpoints in the ground truth path $P = \{v_1, v_2, \ldots, v_{end}\}$.

To compute G_i, we first build a map from each viewpoint v_i in P to a specific sub-instruction in I. The map function is very straightforward: we choose the first sub-instruction I_{sub_i} in I that corresponds to v_i as the mapped sub-instruction. By doing so, each viewpoint v in P now has their own related sub-instruction I_{sub_i} in I. We then compute $G_i = [g_1, ..., g_L]$, by the following steps:

- Find the viewpoint v_i where the agent stands at time step i. If $v_i \notin P$, choose the viewpoint in P that is closest to v_i as the new $v_i \in P$.
- Compute each $g_j \in G_i$ by:

$$g_j = \begin{cases} 1 & \text{if } w_j \in I_{sub_i}, \\ 0.5 & \text{if } w_j \in I_{sub_{i+1}}, \\ -1 & \text{otherwise,} \end{cases} \tag{3}$$

  since every v_i has its mapped I_{sub_i}.

We compute each $\mathcal{L}_{\text{attention}}^{(t)}$ and the total loss becomes:

$$\mathcal{L} = \mathcal{L}_{\text{original}} + \alpha \sum_{t=1}^{T} \mathcal{L}_{\text{attention}^{(t)}}, \tag{4}$$

where $\alpha = 0.5$ is a hyper-parameter and T is the total number of time steps.

C HAMT Model

The HAMT model is based on a large cross-modal transformer encoder on three types of features: text features $X = [cls, w_1, ..., w_L]$, history features $H_t = [h_{cls}, h_1, ..., h_{t-1}]$ and observation features $O_t = [o_1, ..., o_k, o_{stop}]$.

The text features are similar to the ones in the RecBERT model. The difference is that the word embedding features are pre-trained by several proxy tasks (Chen et al., 2021b) instead of by the PREVALENT model.

The history features H_t are obtained from the panoramic views in the previous steps which keep track of the visual and action information in the past. In general, the $[h_{cls}, h_1, ..., h_{t-1}]$ represents the visual and action history information of the navigation in the previous steps.

Different from the candidate features f_C in RecBERT, the observation features O_t contain features of all views from the current viewpoint v_i. To indicate whether there is a navigable viewpoint in the particular view, a "navigable embedding" is added to the observation features to tell the model that such a view leads to a navigable viewpoint.

At time step t, the instruction I is converted to the pre-trained word embeddings X by a multi-layer transformer (which could also be loaded from the last step, if possible). The panoramic view is passed to a vision transformer that outputs the observation feature O_t. History features H_t are computed based on the panoramic views from the previous time steps using transformers. Before features are sent to the cross-modal transformer encoder, H_t and O_t are first concatenated as $[H_t; O_t]$. Inside the cross-modal transformer encoder, the cross-attention and self-attention are computed sequentially on X and $[H_t; O_t]$. In the end, the model produces the encoded results H'_t and O'_t.

To decide which action to take, the HAMT model computes the element-wise product between the cls token in X', which is X'_{cls} and those view features that contain navigable viewpoints from $O_{nav} = [o'_1, ..., o'_n] \in O'_t$: $X'_{cls} \odot O_{nav}$. Two fully-connected layers are used after the element-wise product, and a softmax computation is performed to obtain the probability of each available action:

$$p(o_i) = \frac{\exp(\mathrm{fc}_1(\mathrm{fc}_2(o'_i \odot x'_{cls})))}{\sum_j^{O_{nav}} \exp(\mathrm{fc}_1(\mathrm{fc}_2((o'_j \odot x'_{cls})))}.$$

The loss function used by HAMT is similar to the RecBERT loss, except the A2C algorithm for reinforcement learning loss is replaced by the A3C algorithm Mnih et al. (2016).

D Ablation Study for M and k of Snapshot Ensemble

To find out the influence of period parameter M and ensemble size parameter k on the performance of snapshot ensemble, we evaluated the performance of snapshot ensembles with different values for M and k using the R2R val_unseen split data. We fixed $k = 3$ and $M = 10$ as the initial setting for the ablation study experiments. We tested $M \in \{5, 10, 15\}$ and $k \in \{3, 4, 5\}$. The results are shown in Tables 6 and 7.

M	TL↓	NE↓	SR↑	SPL↑
5	12	3.87	64.58	58.32
10	11.79	3.77	65.18	58.88
15	12.08	3.73	65.3	58.88

Table 6: The ablation study experiment for the number of snapshots to save M. Here we fixed $k = 3$. We saw an improvement when M increases from 5 to 10 (0.40 pp in SR) but a minor improvement from 10 to 15 (0.12 pp in SR).

M	TL↓	NE↓	SR↑	SPL↑
3	11.79	3.77	65.18	58.88
4	11.8	3.75	65.56	59.2
5	11.88	3.8	65.69	59.36

Table 7: The ablation study experiment for the maximum number of snapshots to be in the ensemble k. Here we fixed $M = 10$. We saw an increase of SR when k increases from 3 to 4 (0.38 pp) but not that much from 4 to 5 (0.13 pp).

According to the results, we see an increase of 0.4 pp in SR from $M = 5$ to $M = 10$, while not that much (0.12 pp) from $M = 10$ to $M = 15$. Considering $M = 15$ takes 50% more ensembles to evaluate, we chose $M = 10$ to be our number of snapshots to save during training.

After fixing $M = 10$, we discovered that the ensemble performance improves by 0.38 pp when k increases from 3 to 4. A much less improvement is seen when k increases from 4 to 5 (0.13%). Since setting $k = 5$ requires another 3,000 MB graphics card memory and extra sets of ensembles for evaluation but with seemingly little improvement, we decided to use $k = 4$ as our number of maximum snapshots in the ensemble during beam search.

E Case Study for RecBERT Snapshot Ensemble

We consider the case of our snapshot ensemble agent navigating in a museum-like environment. The panoramic views and model scores are given in Figure 10

The instruction is "Go through the large wooden doors and turn right. Pass the photos on the left and pass the second set of wooden doors. Continue going straight and stop at the chair at the end of the table." In most time steps, we can see that all

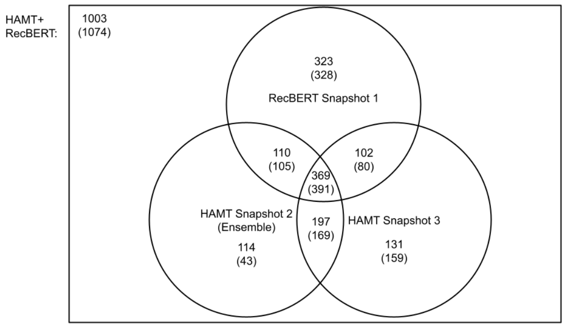
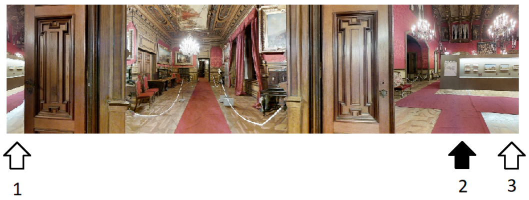


Figure 9: The Venn diagram on val_unseen for the mixed snapshot ensemble of the RecBERT and HAMT models. The pattern is similar to the one in figure 5, showing that the ensemble makes recommendations that are more often equal to those of its members than the members' recommendations are to each other.

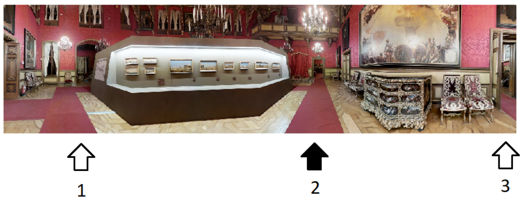
snapshots contribute to deciding what the ensemble should act next. However, exceptions exist. In time step $t = 2$, snapshots 1 and 3 both ignored “turn right" and voted to take action 1. As the only correct snapshot among three, snapshot 2 “forced" the ensemble to take action 2 by predicting the action with a much higher prediction score. This observation suggests that the weighted voting mechanism helps improve the ensemble performance compared to that of its member snapshots.

F Additional Analysis

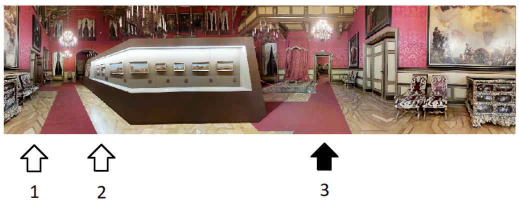
We here show the Venn diagram for the size-3 mixed snapshot ensemble of the RecBERT and HAMT models in Figure 9. The ensemble agent understands and reacts to the instructions in a “more robust way," making less diverse decisions to its snapshots than its snapshots to each other.



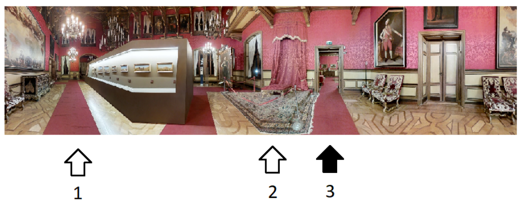
t = 1	action 1	action 2	action 3	stop
ensemble	2.30	**9.70**	7.58	-21.79
snapshot 1	0.99	2.47	2.01	-6.28
snapshot 2	0.05	2.72	3.00	-6.07
snapshot 3	1.26	4.51	2.56	-9.44



t = 2	action 1	action 2	action 3	stop
ensemble	1.80	**7.28**	-26.59	-31.37
snapshot 1	2.80	0.78	-8.30	-8.22
snapshot 2	-2.07	6.56	-7.84	-9.43
snapshot 3	1.06	-0.06	-10.47	-13.73



t = 3	action 1	action 2	action 3	stop
ensemble	-15.10	-16.91	**13.90**	-26.29
snapshot 1	-3.94	-4.77	4.19	-8.01
snapshot 2	-9.26	-10.11	7.49	-9.46
snapshot 3	-1.89	-2.02	2.22	-8.81



t = 4	action 1	action 2	action 3	stop
ensemble	-15.30	5.16	**11.18**	-15.64
snapshot 1	-4.28	1.51	3.07	-5.26
snapshot 2	-10.33	3.02	7.03	-7.53
snapshot 3	-0.69	0.63	1.08	-2.85

t = 5	action 1	action 2	action 3	stop
ensemble	**12.22**	-17.47	-7.45	2.22
snapshot 1	2.48	-4.08	-1.67	0.03
snapshot 2	7.17	-9.66	-3.18	1.90
snapshot 3	2.58	-3.72	-2.60	0.29

t = 6	action 1	action 2	action 3	stop
ensemble	**11.56**	-17.18	-16.52	-1.10
snapshot 1	3.64	-4.83	-4.50	-0.56
snapshot 2	6.65	-9.24	-8.48	1.25
snapshot 3	1.28	-3.12	-3.54	-1.79

t = 7	action 1	action 2	action 3	stop
ensemble	5.98	-0.92	-9.70	**17.39**
snapshot 1	0.25	-1.11	-1.54	3.48
snapshot 2	4.91	1.01	-8.19	5.47
snapshot 3	0.82	-0.83	0.03	8.43

Figure 10: The navigation instruction of this case study is "Go through the large wooden doors and turn right. Pass the photos on the left and pass the second set of wooden doors. Continue going straight and stop at the chair at the end of the table." Left: Panoramic views at each viewpoint. Right: Prediction scores of the ensemble and each snapshot taking action 1, 2, 3, or stop in the current time step. The arrows below the panoramic views point out the directions of the recommended actions with the ensemble action in bold.