

The Regretful Agent: Heuristic-Aided Navigation through Progress Estimation

Chih-Yao Ma^{*†}, Zuxuan Wu[‡], Ghassan AlRegib[†], Caiming Xiong[§], Zsolt Kira[†]

[†]Georgia Institute of Technology, [‡]University of Maryland, College Park, [§]Salesforce Research

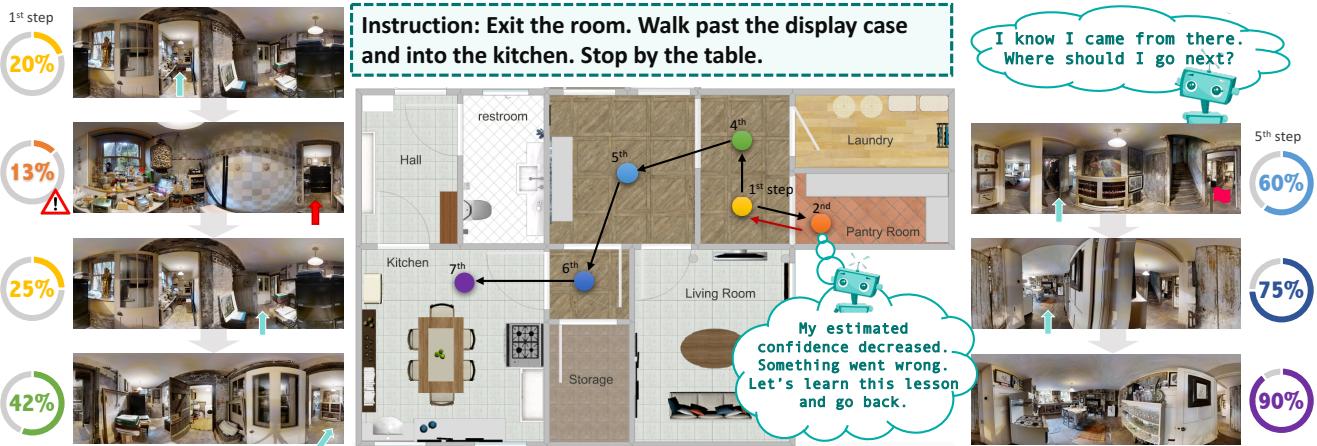


Figure 1: Vision-and-Language Navigation task and our proposed regretful navigation agent. The agent leverages the self-monitoring mechanism [14] through time to decide when to roll back to a previous location and resume the instruction-following task. Our code is available at <https://github.com/chihyaoma/regretful-agent>.

Abstract

As deep learning continues to make progress for challenging perception tasks, there is increased interest in combining vision, language, and decision-making. Specifically, the Vision and Language Navigation (VLN) task involves navigating to a goal purely from language instructions and visual information without explicit knowledge of the goal. Recent successful approaches have made in-roads in achieving good success rates for this task but rely on beam search, which thoroughly explores a large number of trajectories and is unrealistic for applications such as robotics. In this paper, inspired by the intuition of viewing the problem as search on a navigation graph, we propose to use a progress monitor developed in prior work as a learnable heuristic for search. We then propose two modules incorporated into an end-to-end architecture: 1) A learned mechanism to perform backtracking, which decides whether to continue moving forward or roll back to a previous state (Regret Module) and 2) A mechanism to help the agent decide which direction to go next by showing directions that are visited and their associated progress estimate (Progress

Marker). Combined, the proposed approach significantly outperforms current state-of-the-art methods using greedy action selection, with 5% absolute improvement on the test server in success rates, and more importantly 8% on success rates normalized by the path length.

1. Introduction

Building on the success of deep learning in solving various computer vision tasks, several new tasks and corresponding benchmarks have been proposed to combine visual perception and decision-making [2, 24, 7, 13, 19, 25, 5]. One such task is the Vision-and-Language Navigation task (VLN), where an agent must navigate to a goal purely from language instructions and visual input without explicit knowledge of the goal. This task has a number of applications, including service robotics where it would be preferable if humans interacted naturally with the robot by instructing it to perform various tasks.

Recently, there have been several approaches proposed to solve this task. The dominant approaches frame the navigation task as a *sequence to sequence* problem [2]. Several enhancements such as synthetic data augmentation [12], pragmatic inference [12], and combinations of model-free and model-based reinforcement learning techniques [21]

^{*} Work partially done while the author was a research intern at Salesforce Research.

have also been proposed. However, current methods are separated into two regimes: those that use beam search and obtain good success rate (with longer trajectory lengths) and those that use greedy action selection (and hence result in very low trajectory lengths) but obtain much lower success rates. In fact, there have recently been new metrics proposed that balance these two objectives [1]. Intuitively, the agent should perform intelligent action selection (akin to best-first search), without exhaustively exploring the search space. For robotics application, for example, the use of beam search is unrealistic as it would require the robot to explore a large number of possible trajectories.

In this paper, we view the process of navigation as *graph search* across the navigation graph and employ two strategies, encoded within the neural network architecture, to enable navigation without the use of beam search. Specifically, we develop: 1) A **Regret Module** that provides a mechanism to allow the agent to *learn when to backtrack* [11, 3] and 2) We propose a **Progress Marker** mechanism that allows the agent to incorporate information from previous visits and reason about such visits and their associated progress estimates towards better action selection.

Specifically, in graph search a heuristic is used to make meaningful progress towards the goal in a manner that avoids exhaustive search but is more effective than naïve greedy search. We therefore build on recent work [14] that developed a *progress monitor* which is a learned mechanism that was used to estimate the progress made towards the goal (with low values meaning progress has not been made and high values meaning the agent is closer to the goal). In that work, however, the focus was on the regularizing effect of the progress monitor as well as its use in beam search. Instead, we use this progress monitor effectively as a *learned heuristic* that can be used to determine directions that are more likely to lead towards the goal during inference.

We use the Progress Marker in two ways. First, we leverage the notion of backtracking, which is prevalent in graph search, by developing a *learned rollback mechanism* that decides whether to go back to the previous location or not (Regret Module). Second, we incorporate a mechanism to allow the agent to use the estimated progress it computed when visiting the viewpoints to choose the next action to perform after it has rolled back (Progress Marker). This allows the agent to know when particular directions have already been visited and the progress they resulted in, which can bias it to not re-visit states unless warranted. We do this by augmenting the visual state vectors with the progress estimates so that the agent can reduce the probability of revisiting such states (again, in a learned manner).

We demonstrate that these learned mechanisms are superior to greedy decoding. Our agent is able to achieve state-of-the-art results among published works both in terms of success rate (when beam search is not used) and more im-

portantly the SPL [1] metric which incorporates path length, owing to our short trajectory lengths. In summary, our contributions include: 1) A graph search perspective on the instruction-based navigation problem, and use of a **learned heuristic** in the form of a progress monitor to effectively explore the navigation graph, 2) an end-to-end trainable **Regret Module** that can learn to decide when to roll back to the previous location given the history of textual and visual grounding observed, 3) a **Progress Marker** that can enable effective backtracking and reduce the probability of going to a visited location accordingly, and 4) state-of-the-art results on the VLN task.

2. Related Work

Vision and language navigation. There are a number of benchmarks and environments for investigating the combination of vision, language, and decision-making. This includes House3D [24], Embodied QA [7], AI2-THOR [13], navigation based agents [17, 22, 15] (including with communication [9]), and the VLN task that we focus on [2]. For tasks that contain only sparse rewards, reinforcement learning approaches exist [21, 27, 8], for example focusing on language grounding through guided feature transformation [27] and development of a neural module approach [8]. Our work, in contrast, focuses on tasks that contain language instructions that can guide the navigation process and has applications such as service robotics. Approaches to this task are dominated by a sequence-to-sequence formulation, beginning with initial work introducing the task [2]. Subsequent methods have used a Speaker-Follower technique to generate synthetic instructions that are used for data augmentation and pragmatic inference [12], as well as the combination of supervised-based and RL-based approaches [21, 20]. Recently, the Self-Monitoring navigation agent was introduced which learns to estimate progress made towards the goal using visual and language co-grounding [14]. Prior work employs beam-search type techniques, though, optimizing for success rate at the expense of trajectory length and reduced applicability to robotics and other domains. Inspired by the latter work, we view the progress monitor as a learned heuristic and combine it with other techniques in graph search, namely backtracking, to use it for *action selection*, which was not a focus of the prior work.

Navigation and learned heuristics. Several works in vision and robotics have explored the intersection of learning and planning. In robotics, planning systems must often explore large search trees for getting from start to goal, and selection of the next state to expand must be done intelligently to reduce computation. Often fixed heuristics (e.g. distance to goal) are used, but these are static, require known goal locations, and are used for optimal A*-style algorithms rather than greedy best-first search, which

is what can be employed on robots when maps are not available [18]. Recently, several learning-based approaches have been developed for such heuristics, including older works that learn residuals for existing heuristics [26], heuristic ranking methods that enable refinement of new ones [23] as well as learning of a heuristic policy in a Markov Decision Process (MDP) formulation to directly optimize search effort by taking into account history and contextual information [4]. In our work, we similarly learn to estimate a heuristic (progress monitor) and use it for action selection, showing that the resulting estimates can generalize to unseen environments. We also develop an architecture to explicitly learn when to backtrack based on this progress monitor (with a Progress Marker to reduce the chance of choosing the same action again after backtracking unless warranted), which further improves navigation performance.

Modern Reinforcement Learning. Modern Reinforcement Learning methods like Asynchronous Advantage Actor Critic (A3C) [16] or Advantage Actor Critic (A2C) methods are related to the baseline Self-Monitoring agent [14] and the proposed Regretful agent. Specifically, the progress monitor in the Self-Monitoring agent (our baseline) is similar to the value function in RL, and the difference between progress marker of a viewpoint and current progress estimation (denote as $\Delta v_{t,k}^{marker}$, see Sec. 4.2) is conceptually similar to the advantage function. However, the advantage function in RL serves as a way to regularize and improve the training of the policy network. We instead associate the $\Delta v_{t,k}^{marker}$ directly to all navigable states, and this $\Delta v_{t,k}^{marker}$ has a direct impact on the agent deciding next action even during inference. While having an accurate value estimate for VLN with dynamic and implicit goals may reduce the need for this formulation, we however believe that this is hardly possible because of the lack of training data. On the other hand, relating to the proposed end-to-end learned regret module, *Leave no Trace* [10] learns a forward and a *reset* policy to reset the environment for preventing the policy entering a non-reversible state. Instead of learning to reset, we learn to rollback to a previous state and continue the navigation task with a policy network that learns to decide a better next step.

3. Baseline

Given natural language instructions, our task is to train an agent to follow these instructions and reach an (unspecified) goal in the environment (see Figure 1 for an example). This requires processing both the instructions and the visual inputs, along with attentional mechanisms to ground them to the current situation. We adapt the recently introduced Self-Monitoring Visual-Textual Co-grounding agent [14] as our baseline. The Self-Monitoring agent consists of two primary components: (1) A visual-textual co-grounding module that grounds to the completed instruction, the next

instruction, and the subsequent navigable directions represented as visual features. (2) A progress monitor that takes the attention weights of grounded instructions as input and estimates the agent’s progress towards completing the instruction. It was shown that such a progress monitor can *regularize* the attentional mechanism (via an additional loss), but the authors did not focus on using the progress estimates for action selection itself. In the following, we briefly introduce the Self-Monitoring agent.

Specifically, a language instruction with L words is represented via embeddings denoted as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$, where \mathbf{x}_l is the feature vector for the l -th word encoded by a Long Short-Term Memory (LSTM) language encoder. Following [14, 12], we use a panoramic view as visual input. At the t -th time step, the agent perceives a set of images at each viewpoint $\mathbf{v}_t = \{\mathbf{v}_{t,1}, \mathbf{v}_{t,2}, \dots, \mathbf{v}_{t,K}\}$, where K is the maximum number of navigable directions, and $\mathbf{v}_{t,k}$ represents the image feature of direction k obtained from an ImageNet pre-trained ResNet-152. It first obtains visual and textual grounded features, $\hat{\mathbf{v}}_t$, and $\hat{\mathbf{x}}_t$, respectively, with hidden states from the last time step \mathbf{h}_{t-1} using soft-attention (see [14] for details). Conditioned on these grounded features and historical context, it then produces the hidden context of the current step \mathbf{h}_t :

$$\mathbf{h}_t, \mathbf{c}_t = LSTM([\hat{\mathbf{x}}_t, \hat{\mathbf{v}}_t, \mathbf{a}_{t-1}], \mathbf{h}_{t-1}, \mathbf{c}_{t-1}),$$

where $[,]$ denotes concatenation and \mathbf{c}_{t-1} denote cell states from the last time step. To further decide where to go next, the current hidden states \mathbf{h}_t are concatenated with grounded instructions $\hat{\mathbf{x}}_t$, yielding a representation that contains historical context and relevant parts of the instructions (for example, corresponding to parts that have just been carried out and those that have to be carried out next), to compute the correlations with visual features for each viewpoint k ($\mathbf{v}_{t,k}$). Formally, action selection is calculated as follows:

$$o_{t,k} = (\mathbf{W}_a[\mathbf{h}_t, \hat{\mathbf{x}}_t])^\top g(\mathbf{v}_{t,k}) \quad \text{and} \quad \mathbf{p}_t = \text{softmax}(\mathbf{o}_t)$$

where \mathbf{W}_a are the learned parameters and $g(\cdot)$ is a Multi-Layer Perceptron (MLP).

Furthermore, we also equip the agent with a progress monitor following [14] to enforce the attention weights of the textual grounding to align with the progress made toward the goal, further regularizing the grounded instructions to be relevant. The progress monitor is optimized such that the agent is required to use the attention distribution of textual grounding to predict the distance from goal. The output of progress monitor p_t^{pm} represents the completeness of instruction-following estimated by the agent.

$$\mathbf{h}_t^{pm} = \sigma(\mathbf{W}_h([\mathbf{h}_{t-1}, \hat{\mathbf{v}}_t]) \otimes \tanh(\mathbf{c}_t))$$

$$p_t^{pm} = \tanh(\mathbf{W}_{pm}([\mathbf{a}_t, \mathbf{h}_t^{pm}]))$$

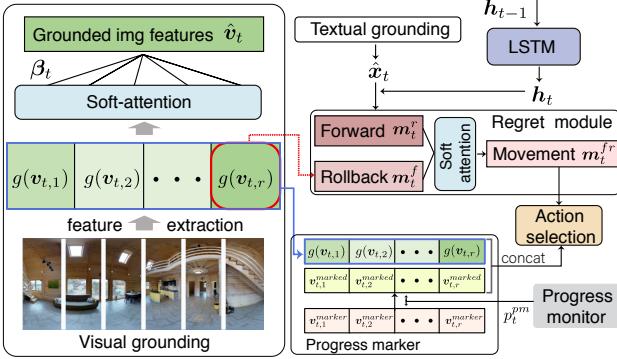


Figure 2: Illustration of the proposed regretful navigation agent. Note that the progress monitor is based on [14].

where W_h and W_{pm} are the learnt parameters, c_t is the cell state of the LSTM, \otimes denotes the element-wise product, α_t is the attention weights of textual grounding, and σ is the sigmoid function. Please refer to [14] for further details on the baseline architecture.

4. Regretful Navigation Agent

The progress monitor previously mentioned reflects the agent’s progress made towards the goal, and consequently its outputs will decrease or fluctuate if the agent selects an action leading to deviation from the goal. Conversely it will increase if it moves closer to the goal by completing the instruction. We posit that such a property, while conceptually simple, provides critical feedback for action selection. To this end, we leverage the outputs of the progress monitor to allow the agent to regret and backtrack using a **Regret Module** and a **Progress Marker** (see Figure. 2). In particular, the Regret Module examines the progress made from the last step to the current step to decide whether to take a *forward* or *rollback* action. Once the agent regrets and rolls back to the previous location, the Progress Marker informs whether location(s) have been visited before and rates the visited location(s) according to the agent’s confidence in completing the instruction-following task. Combining the two proposed methods, we show that the agent is able to perform a local search on the navigational graph by (1) assessing the current progress, (2) deciding when to roll back, and (3) selecting the next location after rollback occurs. In the following, we elaborate these two components in detail.

4.1. Regret Module

The **Regret Module** takes in the outputs of the progress monitor at different time steps and decides whether to go *forward* or to *rollback*. In particular, we use the concatenation of hidden state h_t and grounded instruction \hat{x}_t as our *forward* embedding \mathbf{m}_t^f , and more importantly we intro-

duce a *rollback* embedding \mathbf{m}_t^r to be the projection of the visual features for the action that leads to the previously visited location. The two vector representations are as follows:

$$\mathbf{m}_t^f = \mathbf{W}_a[\mathbf{h}_t, \hat{\mathbf{x}}_t] \quad \text{and} \quad \mathbf{m}_t^r = g(\mathbf{v}_{t,r}),$$

where \mathbf{W}_a are the learned parameters, $\hat{\mathbf{x}}_t$ is the grounded instruction obtained from the textual grounding module, and $\mathbf{v}_{t,r}$ is the image feature vector representing a direction that points to the previously visited location.

To decide whether to go forward or rollback, the Regret Module leverages the difference of the progress monitor outputs between the current time step and the previous time step $\Delta p_t^{pm} = p_t^{pm} - p_{t-1}^{pm}$. Intuitively, if the difference is larger than a certain threshold $\Delta p_t^{pm} > \sigma$, the agent should decide to take a *forward* action, and vice versa. Since it is hard to decide an optimal value for σ , we achieve this by computing attention weights α_t^{fr} and perform a weighted sum on both *forward* and *rollback* embeddings. If the weight on *rollback* is larger, the agent is likely to be biased to take an action that leads to the last visited location. Formally, the weights can be computed as:

$$\begin{aligned} \alpha_t^{fr} &= \text{softmax}(\mathbf{W}_r(\Delta p_t^{pm})) \\ \mathbf{m}_t^{fr} &= (\alpha_t^{fr})^\top [\mathbf{m}_t^f, \mathbf{m}_t^r], \end{aligned}$$

where \mathbf{W}_r are the learnt parameters, $[,]$ denotes concatenation between feature vectors, and \mathbf{m}_t^{fr} represents the weighted sum of the *forward* and *rollback* embeddings. Note that to ensure the progress monitor remains focused on estimating the agent’s progress and regularizing the textual grounding module, we *detach* the output of the progress monitor which is fed into the Regret Module and set it as a leaf in the computational graph.

Action selection. Similar to existing work, the agent determines which image features from navigable directions have the highest correlation with the movement vector \mathbf{m}_t^{fr} by computing the inner-product, and the probability of each navigable direction is then computed as:

$$\mathbf{o}_{t,k} = (\mathbf{W}_{fr}\mathbf{m}_t^{fr})^\top g(\mathbf{v}_{t,k}) \quad \text{and} \quad \mathbf{p}_t = \text{softmax}(\mathbf{o}_t),$$

where \mathbf{W}_{fr} are the learned parameters and \mathbf{p}_t is the probability distribution over navigable directions at time t . In practice, once the agent takes a rollback action, we block the action that leads to oscillation.

4.2. Progress Marker

The Regret Module provides a mechanism for the agent to decide when to rollback to a previous location or move forward according to the progress monitor outputs. Once the agent rolls back, it is required to select the next direction to go forward. It is thus essential for the agent to (1) know which directions it has already visited (and rolled back) and

(2) estimate if the visited locations can lead to a path which completes the given instruction.

Toward this end, we propose the **Progress Marker** to mark each visited location with the agent’s confidence in completing the instruction (see Figure 3). More specifically, we maintain a set of memory M and store the output of the progress monitor associated with each visited location; if the location is not yet visited, the marker will be filled with a value 1:

$$\mathbf{v}_{t,k}^{\text{marker}} = \begin{cases} p_i^{\text{pm}}, & \text{if } k \text{ leads to a location } i \in M. \\ 1, & \text{otherwise.} \end{cases}$$

where i is a unique viewpoint ID for each location. We allow the marker on each location to be updated every time the agent visits it.

The marker value on each navigable direction indicates the estimated confidence that a location leads to the goal. We assign a value 1 for unvisited directions to encourage the agent to explore the environment. The navigating probabilities between unvisited directions depend on the action probabilities p_t since their marker values are the same.

Action selection with Progress Marker. During action selection, in addition to the movement vector \mathbf{m}_t^{fr} that the agent can rely on in deciding which direction to go, we propose to label the marker value to each navigation direction as indications of whether a direction is likely to lead to the goal or to unexplored (and potentially better) paths. To achieve this, we leverage the difference between the current estimated progress and the marker for each navigable direction $\Delta v_{t,k}^{\text{marker}} = p_t^{\text{pm}} - v_{t,k}^{\text{marker}}$. We then concatenate it to the visual feature representation for each navigable direction before action selection.

$$\mathbf{v}_{t,k}^{\text{marked}} = [g(\mathbf{v}_{t,k}), \Delta v_{t,k}^{\text{marker}}].$$

The difference $\Delta v_{t,k}^{\text{marker}}$ indicates the chances of navigable directions leading to the goal and further inform the agent which direction to select. In our design, lower $\Delta v_{t,k}^{\text{marker}}$ corresponds to higher chance for action selection. For instance, in step 4 in Figure 3, the $\Delta v_{t,k}^{\text{marker}}$ for starting location and the last visited location are 0.08 and -0.02 respectively, whereas an unvisited location will have -0.71, which eventually leads to 0.52 estimated progress.

When using Progress Marker, the final action selection is formulated as:

$$\mathbf{o}_{t,k} = (\mathbf{W}_{\text{fr}} \mathbf{m}_t^{\text{fr}})^{\top} \mathbf{v}_{t,k}^{\text{marked}} \quad \text{and} \quad \mathbf{p}_t = \text{softmax}(\mathbf{o}_t)$$

In practice, we tiled the difference n times before concatenating with the projected image feature $\mathbf{v}_{t,k}$ in order to account for imbalance. The marker value for the *stop* action is set to be 0.

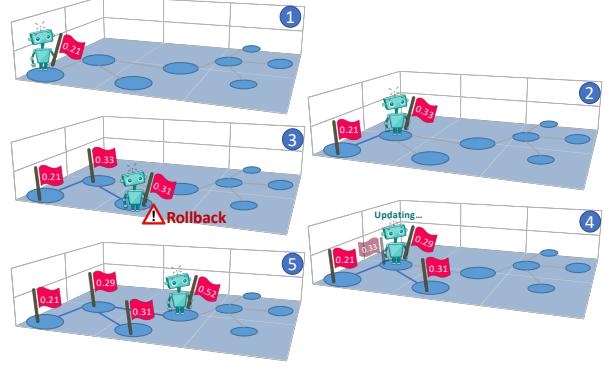


Figure 3: Concept of the proposed Progress Marker (red flags). The agent marks each visited location with estimated progress made towards the goal. The changes on the estimated progress determines whether the agent should *rollback* or *forward*, and the difference between the current estimated progress and the markers on the next navigable directions helps the agent decide which direction to go.

4.3. Training and Inference

We train the proposed agent with cross-entropy loss for action selection and Mean Squared Error (MSE) loss for progress monitor. In addition to these losses, we also introduce an additional entropy loss to encourage the agent to explore other actions, such that it is not biased to actions with already very high confidence. The motivation is that, after training an agent for a period of time, the agent starts to overfit and perform fairly well on the training set. As a result, the agent will not learn to properly roll back during training since the majority of the training samples do not require the agent to roll back. Introducing the entropy loss increases the chance of exploration and making incorrect actions during training.

$$\mathcal{L}_{\text{loss}} = \lambda \underbrace{\sum_{t=1}^T y_t^{\text{nv}} \log(p_{t,k})}_{\text{action selection}} + (1 - \lambda) \underbrace{\sum_{t=1}^T (y_t^{\text{pm}} - p_t^{\text{pm}})^2}_{\text{progress monitor}} - \beta \underbrace{\sum_{t=1}^T \sum_{k=1}^K -p_{t,k} \log(p_{t,k})}_{\text{entropy loss}},$$

where $p_{t,k}$ is the action probability of each navigable direction, y_t^{nv} is the ground-truth navigable direction at step t , $\lambda = 0.5$ is the weight balancing the cross-entropy loss and MSE loss, and $\beta = 0.01$ is the weight for entropy loss.

Following existing approaches [14, 12, 2], we perform categorical sampling during training for action selection. During inference, the agent greedily selects the action with highest action probability.

Table 1: Comparison with the state of the arts with greedy decoding for action selections¹. *: with data augmentation.

Method	Validation-Seen				Validation-Unseen				Test (unseen)			
	NE ↓	SR ↑	OSR ↑	SPL ↑	NE ↓	SR ↑	OSR ↑	SPL ↑	NE ↓	SR ↑	OSR ↑	SPL ↑
Random	9.45	0.16	0.21	-	9.23	0.16	0.22	-	9.77	0.13	0.18	0.12
Student-forcing [2]	6.01	0.39	0.53	-	7.81	0.22	0.28	-	7.85	0.20	0.27	0.18
RPA [21]	5.56	0.43	0.53	-	7.65	0.25	0.32	-	7.53	0.25	0.33	0.23
Speaker-Follower [12]*	3.36	0.66	0.74	-	6.62	0.36	0.45	-	6.62	0.35	0.44	0.28
RCM [20]*	3.37	0.67	0.77	-	5.88	0.43	0.52	-	6.01	0.43	0.51	0.35
Self-Monitoring [14]*	3.22	0.67	0.78	0.58	5.52	0.45	0.56	0.32	5.99	0.43	0.55	0.32
Regretful	3.69	0.65	0.72	0.59	5.36	0.48	0.61	0.37	-	-	-	-
Regretful*	3.23	0.69	0.77	0.63	5.32	0.50	0.59	0.41	5.69	0.48	0.56	0.40

5. Dataset and Implementations

Room-to-Room dataset. We use the Room-to-Room (R2R) dataset [2] for evaluating our proposed approach. The R2R dataset is built upon the Matterport3D dataset [6]. It consists of 10,800 panoramic views from 194,400 RGB-D images in 90 buildings and has 7,189 paths sampled from its navigation graphs. Each path has three ground-truth navigation instructions written by humans. The whole dataset has 90 scenes: 61 for training and validation seen, 11 for validation unseen, 18 for test unseen.

Evaluation metrics. To compare to existing work, we show the same evaluation metrics used in those works: (1) Navigation Error (NE), mean of the shortest path distance in meters between the agent’s final position and the goal location. (2) Success Rate (SR), the percentage of final positions less than 3m away from the goal location. (3) Oracle Success Rate (OSR), the success rate if the agent can stop at the closest point to the goal along its trajectory. However, we note the importance of a recently added metric that emphasizes the trade-off between success rate and trajectory length: Success rate weighted by (normalized inverse) Path Length (SPL) [1], which incorporates trajectory lengths and is an important consideration for real-world applications such as robotics.

Implementation Details. For fair comparison with existing work, we use the pre-trained ResNet-152 on ImageNet to extract image features. Following the Self-Monitoring [14] and Speaker-Follower [12] works, the embedded feature vector for each navigable direction is obtained by concatenating an appearance feature with a 4-d orientation feature $[sin\phi; cos\phi; sin\theta; cos\theta]$, where ϕ and θ are the heading and elevation angles. Please refer to the Appendix for further implementation details.

¹Note that both Speaker-Follower [12] and Self-Monitoring [14] were originally designed to optimize the success rate (SR) via beam search, and concurrently to our work, RCM [20] proposed a new setting allowing the agent to explore unseen environments prior to the navigation task via Self-Supervised Imitation Learning (SIL).

6. Evaluation

6.1. Comparison with Prior Art

We first compare the proposed regretful navigation agent with the state-of-the-art methods [14, 12, 20]. As shown in Table 1, our method achieves significant performance improvement over the existing approaches. We achieved 37% SPL and 48% SR on the validation unseen set and outperformed all existing work. Our best performing model achieves 41% SPL and 50% SR on validation unseen set when trained with the synthetic data from the Speaker [12]. We demonstrate absolute 8% SPL improvement and 5% SR improvement on the test server over the current state-of-the-art method. We can also see that our regretful navigation agent without data augmentation has already outperformed the existing work on both SR and SPL metrics.

6.2. Ablation Study

Table 2 shows an ablation study to analyze the effect of each component. The first thing to note is that our method is significantly better than the Self-Monitoring agent which uses greedy decoding, even though it still has a progress monitor loss (although the progress monitor is not used for action selection). A second interesting point is that when the Progress Marker is available with the features of each navigable direction that have been visited before, but the Regret Module is not available, performance does not increase significantly (44% SR). Note that we also conducted an experiment with another condition, where the progress monitor estimates were attached to the *forward* embedding, meaning that the network could use that information to improve action selection. That condition again was only able to achieve modest gains (45% SR), compared to our Regret Module which was able to achieve 47% SR (and 48% when the Progress Marker was added). In all, this shows that the key improvement stems from the design of the Regret Module, allowing the agent to intelligently backtrack after making mistakes.

Table 2: Ablation study showing the effect of each proposed components compared to the prior arts. All methods here trained without data augmentation.

Method	#	Regret Module	Progress Marker	Validation-Seen				Validation-Unseen			
				NE ↓	SR ↑	OSR ↑	SPL ↑	NE ↓	SR ↑	OSR ↑	SPL ↑
Speaker-Follower [12]				4.86	0.52	0.63	-	7.07	0.31	0.41	-
Self-Monitoring [14]				3.72	0.63	0.75	0.56	5.98	0.44	0.58	0.30
Regretful	1	✓		3.88	0.64	0.70	0.58	5.65	0.47	0.59	0.37
	2		✓	3.76	0.63	0.73	0.57	5.74	0.44	0.59	0.32
	3	✓	✓	3.69	0.65	0.72	0.59	5.36	0.48	0.61	0.37

Table 3: Sanity check for verifying that the source of performance improvement is from the agent’s ability to decide when to roll back.

Method	Blocking Rollback	Validation-Seen				Validation-Unseen			
		NE ↓	SR ↑	OSR ↑	SPL ↑	NE ↓	SR ↑	OSR ↑	SPL ↑
Self-Monitoring [14]		3.72	0.63	0.75	0.56	5.98	0.44	0.58	0.30
	✓	3.85	0.64	0.75	0.58	6.02	0.44	0.60	0.34
Regretful		3.69	0.65	0.72	0.59	5.36	0.48	0.61	0.37
	✓	3.91	0.64	0.68	0.60	5.80	0.46	0.55	0.41

6.3. Rollback Analysis

We now further analyze the behavior of the agent to verify that the source of improvement is indeed from the ability to learn when to roll back.

Does rollback lead to the performance improvement?

Our proposed regretful agent relies on the ability to regret and roll back to a previous location, further exploring the unknown environment to increase the success rate. As a sanity check, we manually block all actions leading to rollback for both the state-of-the-art Self-Monitoring agent and our proposed agent². The result is shown in Table 3. As can be seen, blocking rollback for the Self-Monitoring agent produces mixed results, with worse NE but better metrics such as OSR. The SR, however, is unchanged. On the other hand, blocking rollback for our agent significantly reduces most metrics including NE, SR, and OSR especially on unseen environments. This shows that blocking the ability to learn when to roll back degrades a large source of performance increase, and this is especially true for unseen environments.

Number of unsuccessful examples reduced. We calculate the total number of unsuccessful examples involves rollback action for both Self-Monitoring and our proposed agent (in percentage). As demonstrated in Figure 4, our proposed regretful agent significantly reduces the unsuccessful examples from around 43% to 38%, which correlates to the 4-5% improvement on SR in Table 1 and 2.

Regretful agent in unfamiliar environments. The key

²except when there is only one navigable direction to go.

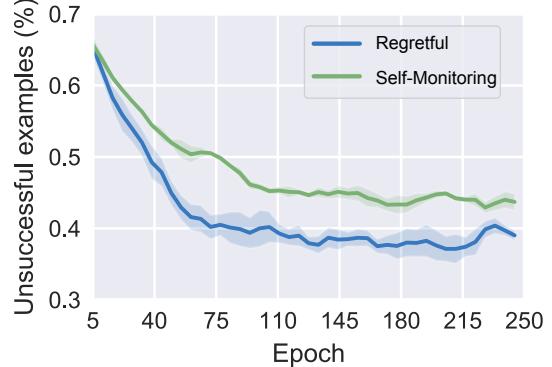


Figure 4: Percentage of unsuccessful examples involving rollback reduced by our proposed regretful agent.

to the performance increase of an agent focusing on the rollback ability is not that the agent learns a better textual or visual grounding, but that the agent learns to search especially when it is not certain which direction to go. To demonstrate this, we train both the Self-Monitoring agent and our proposed regretful agent only on synthetic data and test them on the unseen validation set (real data). We expect the regretful agent to outperform the Self-Monitoring agent across all metrics since our agent is designed to operate in an environment where the agent is likely to be uncertain on action selection. As shown in Table 4, when trained using only the synthetic data, our method significantly outperformed Self-Monitoring agent. Interestingly, when compared with the Self-Monitoring agent trained with

Table 4: Ablation study when trained using only the synthetic or real training data. Oracle Navigation Error (ONE): the navigation error if the agent can stop at the closest point to the goal along its trajectory.

Method	Synthetic	Real	Validation-Unseen		
			ONE ↓	SR ↑	OSR ↑
Self-Monitoring [14]	✓		4.09	0.35	0.49
		✓	3.62	0.44	0.58
Regretful	✓		3.47	0.41	0.58
		✓	3.34	0.48	0.61

real data, our agent trained with synthetic data is slightly better on ONE, same on OSR, and marginally lower on SR. We achieved slightly better performance on oracle metrics since stopping at the correct location is not a hard constrain. This indicates that even though our regretful agent is not yet learned how to properly stop at the goal (due to training on synthetic data only), the chance that it passes/reaches the goal is slightly higher than Self-Monitoring agent trained with real data. Further, when the regretful agent trained with real data, the performance improved across all metrics.

6.4. Qualitative Results

Figures 5 show qualitative outputs of our model during successful navigation in unseen environments. In Figure 5 (a), the agent made a mistake at the first step, and the estimated progress at the second step slightly decreases. The agent then decides to rollback, after which the progress monitor significantly increases. Finally, the agent stopped correctly as instructed. Figure 5 (b) shows an example where the agent correctly goes up the stairs but incorrectly does it again rather than turning and finding the TV as instructed. Note that the progress monitor increases but only by a small amount; this demonstrates the need for learned mechanisms that can reason about the textual and visual grounding and context, as well as the resulting level of change in progress. In this case the agent then correctly decides to rollback and successfully walked into the TV room. Similarly, in Figure 5 (c), the agent misses the stairs, resulting in a very small progress increase. The agent decides to rollback as a result. Upon reaching the goal, the agent’s progress estimate is 99%. Please refer to the Appendix for the full trajectories and unsuccessful examples.

7. Conclusion

In this paper, we have proposed a end-to-end trainable regretful navigation agent for the VLN task. Inspired by the intuition of viewing this task as graph search over the navigation graph, we specifically use a progress monitor as a learned heuristic that can be trained and employed during inference to greedily select the next best action (best-first search). The progress monitor incorporates informa-

tion from grounded language instructions and visual information, integrated across time with LSTMs. We then propose a Regret Module that is able to learn to decide when to perform backtracking depending on the progress made and state of the agent. Finally, a Progress Marker is used to allow the agent to reason about previous visits and unvisited directions, so that the agent can choose a better navigable direction by reducing action probabilities for visited locations with lower progress estimate.

The resulting framework is able to achieve state-of-the-art success rates compared to existing published methods on the public leaderboard, without using beam search. We show through several extensive analyses that the source of performance improvement is the design of the learned rollback mechanism, that when blocked the performance decreases, and that this learning can occur even on purely synthetic data and generalize to real data. We also demonstrated that the total number of unsuccessful examples involve rollback reduces with our regretful agent. There is a great deal of future work possible, which extends our method. For example, other aspects of graph search can be incorporated such as elements of exploration (e.g. a search space frontier), but in a manner that is more efficient than beam search, can also be investigated. Finally, a combination of goal-driven perception and reinforcement learning would be interesting to explore, as the tasks contain less and less structured information (e.g. embodied QA).

Acknowledgments

This research was partially supported by DARPA’s Life-long Learning Machines (L2M) program, under Cooperative Agreement HR0011-18-2-001. We thank Chia-Jung Hsu for her valuable and artistic suggestions on the figures.

Appendix

A. Network Architecture

The embedding dimension of the instruction encoder is 256, followed by a dropout layer with ratio 0.5. We encode the instruction using a regular LSTM, and the hidden state is 512 dimensional. The MLP g used for projecting the raw image feature is $BN \rightarrow FC \rightarrow BN \rightarrow Dropout \rightarrow ReLU$. The FC layer projects the 2176-d input vector to a 1024-d vector, and the dropout ratio is set to be 0.5. The hidden state of the LSTM which allows integration of information across time is 512. When using the progress marker, the markers are tiled $n = 32$ times. The dimension of the learnable matrices are: $\mathbf{W}_x \in \mathbb{R}^{512 \times 512}$, $\mathbf{W}_v \in \mathbb{R}^{512 \times 1024}$, $\mathbf{W}_a \in \mathbb{R}^{1024 \times 1024}$, $\mathbf{W}_r \in \mathbb{R}^{1 \times 2}$, $\mathbf{W}_{fr} \in \mathbb{R}^{1024 \times 1024}$ without progress marker, and $\mathbf{W}_{fr} \in \mathbb{R}^{1024 \times 1056}$ with progress marker.

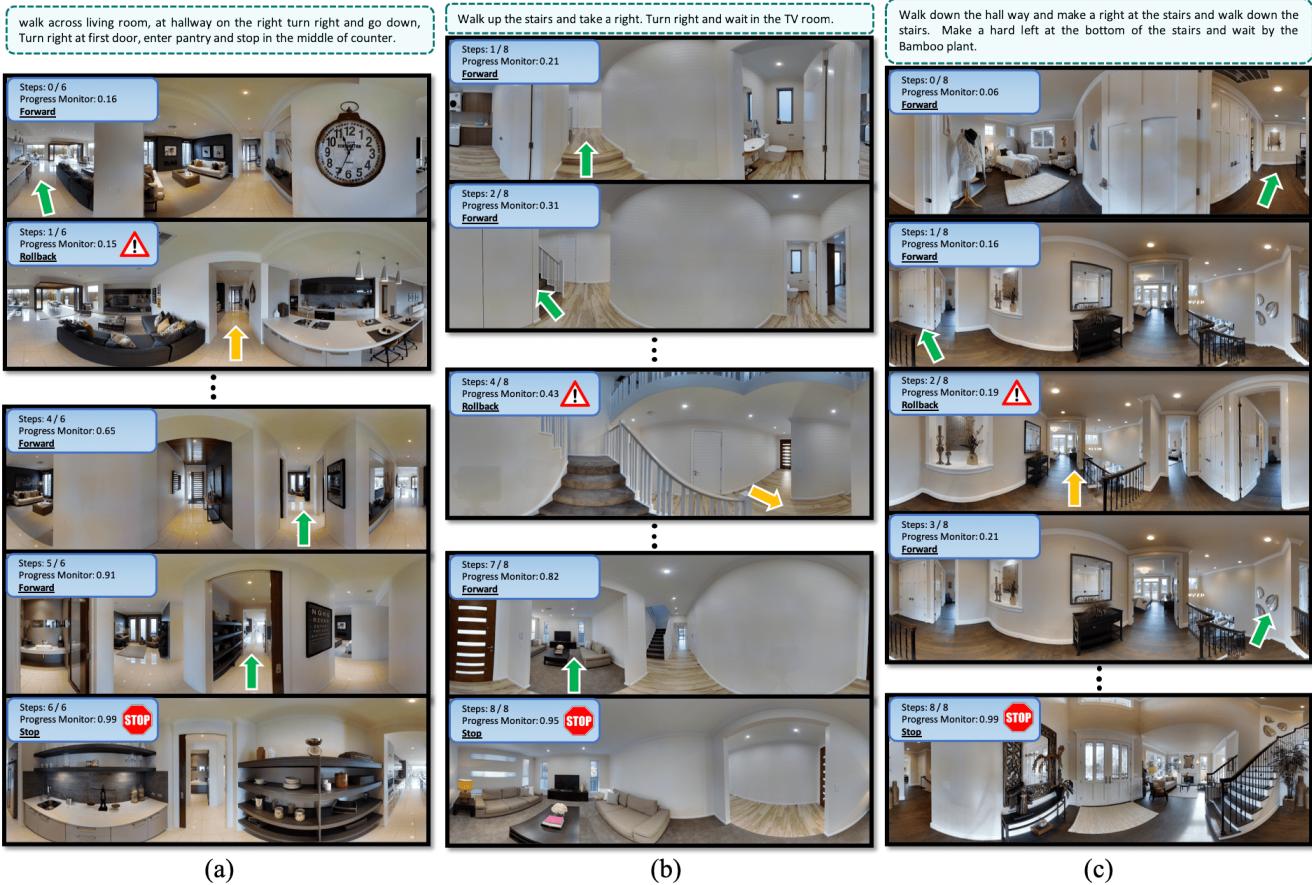


Figure 5: Successful regretful agent navigates in unseen environments. (a) The agent made a mistake at the first step, but it was able to roll back to the previous location since the output of the progress monitor was not significantly increased. It then follows the rest of the instruction correctly. (b) The agent is able to correctly follow the instruction at the beginning but made a mistake by walking up the stairs again. The agent realized that the output of the progress monitor is decreased and the next action *take a right* is not feasible and decides to rollback at step 4. The agent was then able to follow the rest of the instruction and stop with estimated progress 0.95. (c) The agent made a mistake by missing the stairs at step 1. It was however able to decide to rollback at step 2 and moves down stairs as instructed and successfully stops near the bamboo plant with estimated progress 0.99. Please see Appendix for the full trajectories.

B. Comparison with Beam Search Methods

We compare our method using greedy action selection with existing beam search approaches, e.g., Pragmatic Inference in Speaker-Follower [12] and progressed integrated beam search in Self-Monitoring agent [14]. We can see in Table 5 that, while beam search methods perform well on success rate (SR), their trajectory lengths are significantly longer, achieving low success rate weighted by Path Length (SPL) scores and therefore are impractical for real-world applications. On the other hand, our proposed method significantly improved both SR and SPL when not using beam search.

Table 5: Comparison of our regretful agent using greedy action selection with beam search.

Method	Beam search	Test set (leaderboard)			
		NE ↓	SR ↑	Length ↓	SPL ↑
Speaker-Follower [12]	△	6.62	0.35	14.82	0.28
Self-Monitoring [14]	△	5.99	0.43	17.11	0.32
Regretful		5.69	0.48	13.69	0.40

C. Qualitative Analysis

C.1. Successful examples

We show the complete trajectory of the agents successfully deciding when to roll back and reach the goal in unseen environments in Figure 6, 7, 8, and 9.

In Figure 6, we demonstrate that the agent is capable of performing a local search on the navigation graph. Specifically, from step 0 to step 3, the agent searched two possible directions and decided to move with one particular direction at step 4. Once it reached step 5, the agent decides to continue to move forward, and we observed that the progress estimate significantly increased to 45% at step 7. Interestingly, unlike other examples we have shown, the agent did not decide to roll back despite the progress estimate slightly decreased from 45% to 40%. We reckon that this is one of the advantages of using a learning-based regret module, where a learned and dynamically changing threshold decides when to rollback. Finally, the agent successfully stopped in front of the microwave.

In Figure 7, the agent is instructed to *walk across living room*. It is ambiguous since both directions seem like a living room. Our agent first decides to move into the direction that leads to a room with a kitchen and living room. It then decided to roll back with the progress monitor output slightly decreased. The agent then followed the rest of the instruction successfully with the progress monitor steadily increased at each step after that. Finally, the agent decides to stop with the progress estimate 99%.

In Figure 8, the agent first moved out of the room and walked up the stairs as instructed, but the second set of stairs makes the instruction ambiguous. The agent continued to walk up the stairs for one more step and then decided to go down the stairs at step 4. As the agent decided to turn right at step 6, we can see the progress estimate significantly increased from 51% to 66%. Once the agent entered the TV room, the progress estimate increased again to 82%. Finally, the agent successfully stopped with the progress monitor output 95%.

In Figure 9, the agent failed to *walk down the stairs* at step 1. Because of the proposed Regret Module and Progress Marker, the agent was able to discover the correct path to go downstairs. Once walking down, the progress estimate increased to 39% immediately, and as the agent goes further down, the progress estimate reached 98% by the time the agent reached the bottom of the stairs. Finally, the agent decided to wait by the bamboo plant with progress estimate 99%.

C.2. Failed examples

We have shown how the agent can successfully utilize the rollback mechanism to reach the goal, even though it is not familiar with the environment and likely to be uncertain about some actions it took. Intuitively, the rollback mechanism can increase the chance that the agent reaches the goal as long as the agent can correctly decide when to stop.

We now discuss two failed examples of our proposed regretful agent in unseen environments that highly resemble the successful examples in terms of the given instruction

and ground-truth path. Both examples demonstrate that the agent successfully rolled back to the correct path towards the goal but failed to stop at the goal.

Specifically, in Figure 10, the agent reaches the room with the white cabinet as instructed but decided to move one step forward. The agent then decided to roll back to the room correctly at step 5. However, this does not help the agent to stop at the goal resulting in a failed run.

On the other hand, in Figure 11, we can see that the progress estimate at step 5 significantly dropped by 21%, and the agent correctly decided to roll back. The agent then successfully reached the refrigerator but did not stop immediately. It continued to move forward after step 8, resulting in an unsuccessful run.

Lastly, we discuss a failed example when the agent incorrectly decided when to roll back. In Figure 12, the agent first followed the instruction to *go down the hallway* and tried to find the second door to turn right. As the agent reached the end of the hallway at step 4, it decided to roll back since there is no available navigable direction that leads to *turn right*. The agent then decided to go down the hallway again with completely opposite direction. However, the agent decided to roll back again at step 7 with the progress estimate dropped to 18%. Although the agent eventually was able to *escape* from the hallway leading to the dead end, it ends up unsuccessful.

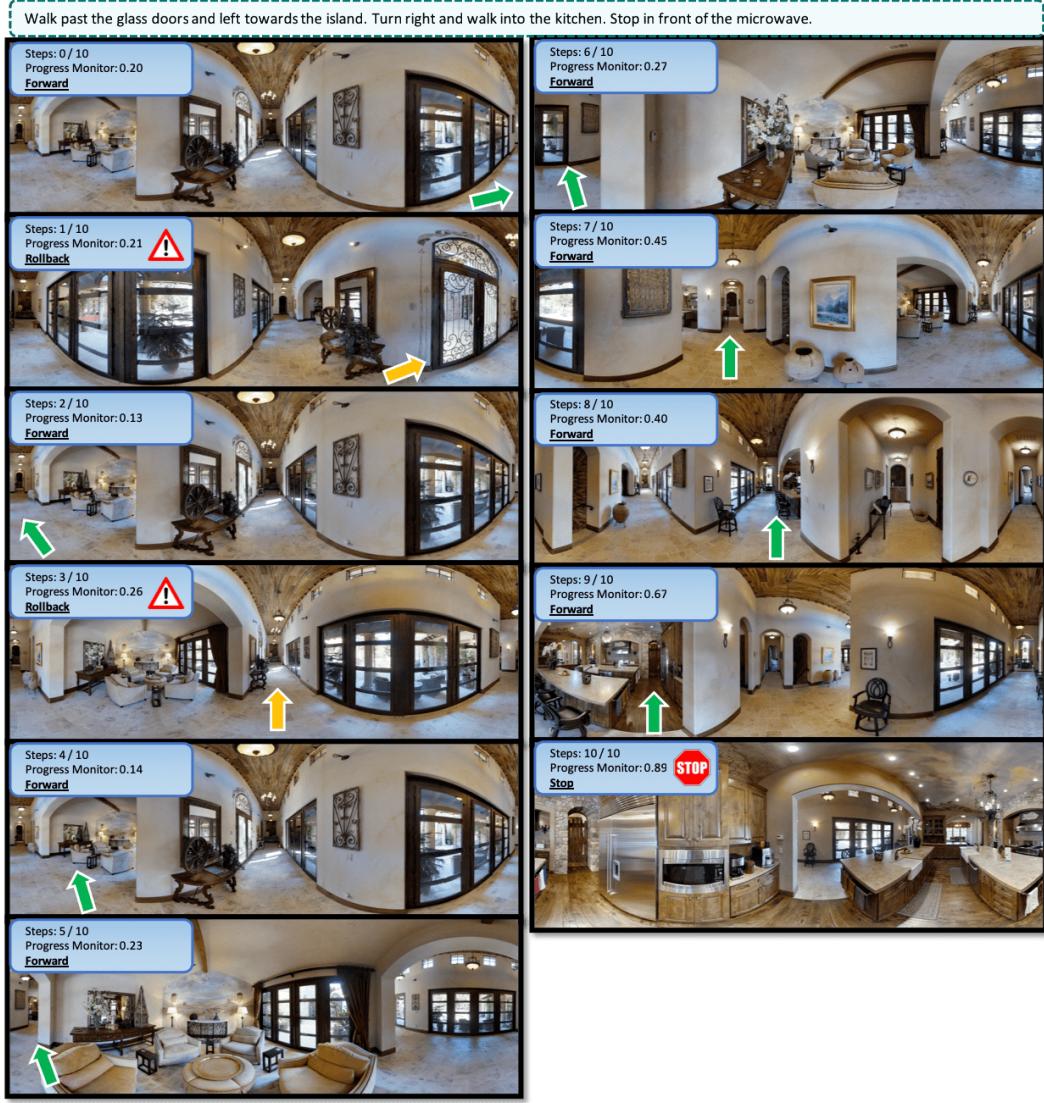


Figure 6: The first part of the instruction *walk past the glass doors* is ambiguous since there are multiple directions that lead to glass doors, and naturally the agent is confused and uncertain where to go. Our agent is able to perform local search on the navigation graph and decides to roll back multiple times at the beginning of the navigation. At step 6, the agent performs an action *turn right*. Consequently, the progress estimate at step 7 significantly increased to 45%. Interestingly, the agent continues to move forward even though the progress estimate slightly decreased from step 7 to step 8. We reckon that this is one of the advantages of using a learning-based regret module as opposed to using a hard-coded threshold. The agent then successfully follows the instruction and *stops in front of the microwave* with progress estimate 89%.



Figure 7: The agent first *walk across living room*, but decides to move into the direction that leads to kitchen and dinning room. At step 1, the agent decides to roll back due to a decreasing of the progress monitor output. The agent then followed the rest of the instruction successfully with the progress monitor steadily increased at each step. Finally, the agent decides to stop with the progress estimate 99%.

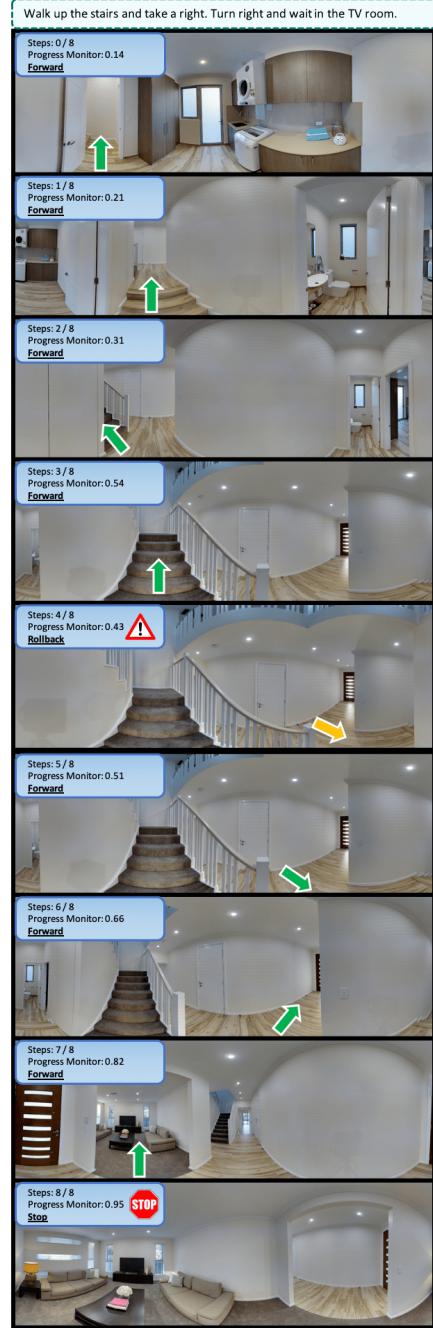


Figure 8: The agent walked up the stairs as instructed at step 1, but the second set of stairs makes the instruction ambiguous. The agent continues to walk up stairs but soon realized that it needs to go down the stairs and *turn right* from step 4 - 6. When the agent decides to turn right, we can see the progress estimate significantly increased from 51% to 66%. As the agent turned right to the TV room, the progress estimate increased again to 82%. Finally, the agent stops with the progress monitor output 95%.

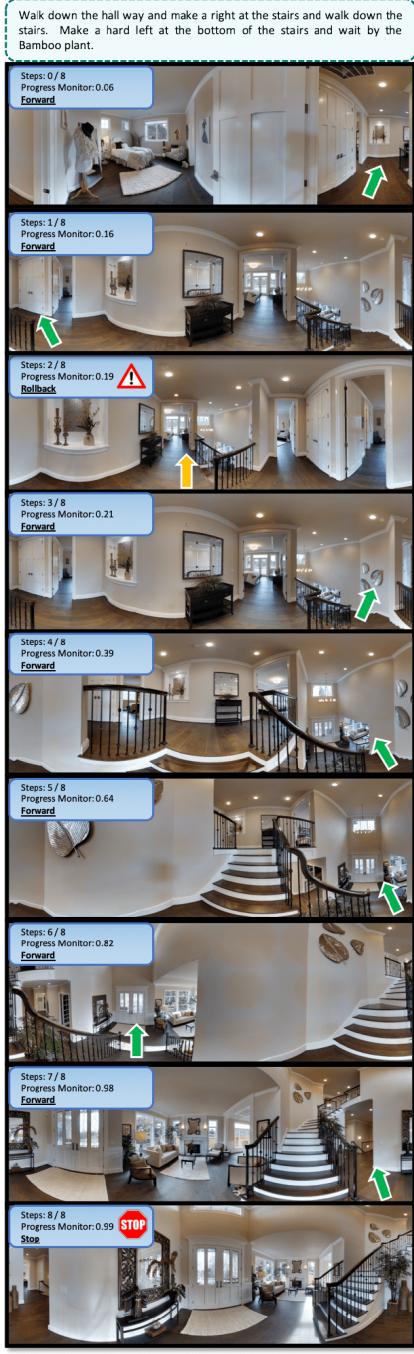


Figure 9: The agent walks down the hall way to the stairs but failed to *walk down the stairs* at step 1. With a small increase on the progress monitor output, the agent then decides to roll back and take the action to walk down the stairs. Once walking down, we can see the progress estimate increased to 39%, and as the agent goes further down, the progress estimate reached 98% at the bottom of the stairs. Finally, the agent decides to stop near by the bamboo plant with progress estimate 99%.



Figure 10: **Failed example.** The agent starts to navigate through the unseen environment by following the given instruction. It was able to successfully follow the instruction and correctly reach the goal at step 4. The agent then decided to move forward towards the kitchen and correctly decided to roll back to the goal. However, the agent did not stop and continue to explore the environment and eventually stopped a bit further from the goal.



Figure 11: The agent correctly followed the first parts of the instruction until step 4, but it decided to move forward towards the hall. At step 5, the agent correctly decided to roll back with the progress estimate decreased from 56% to 35%. The agent was then able to follow the rest of the instruction successfully and reach the refrigerator at step 8. However, the agent did not stop nearby the refrigerator and continued to take another two forward steps.

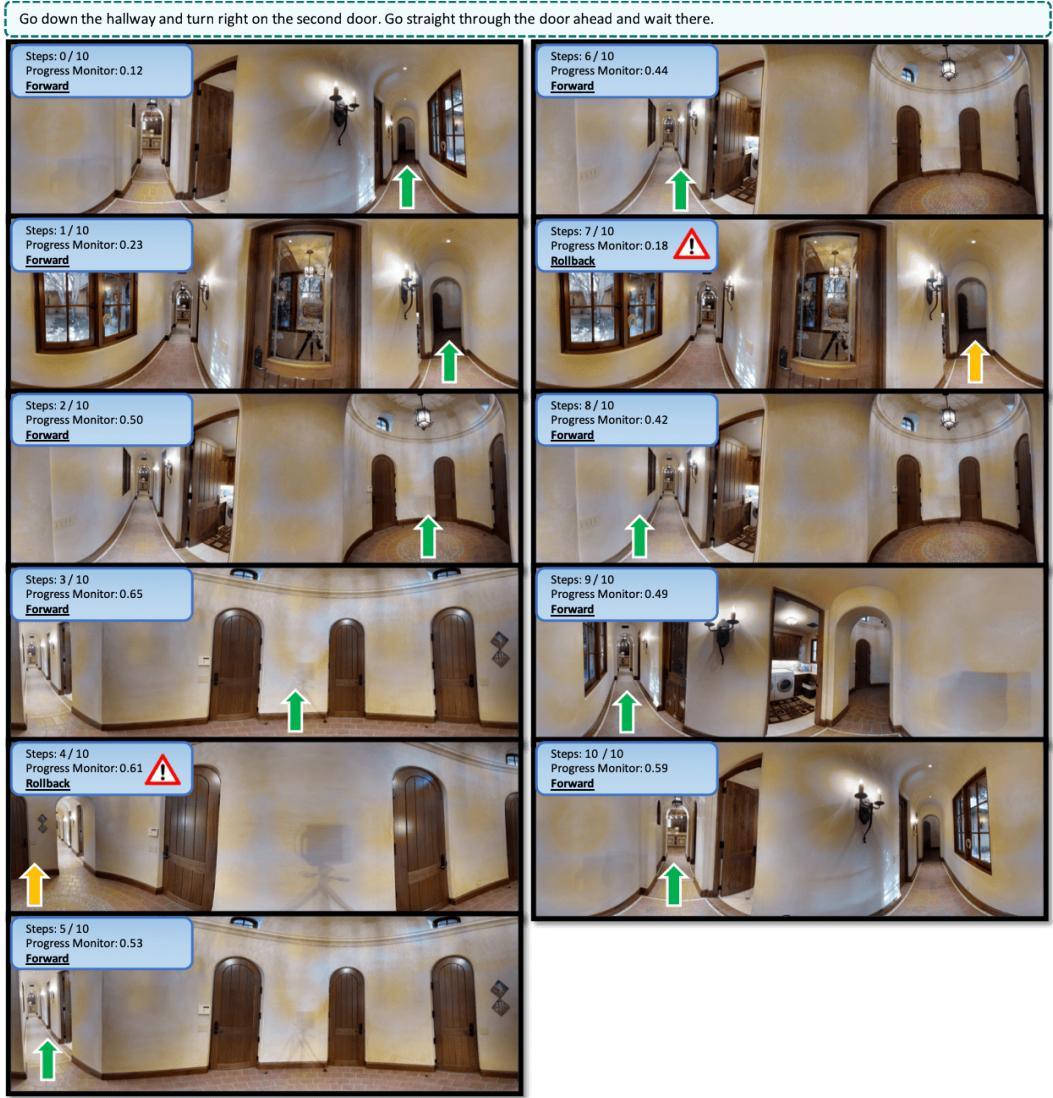


Figure 12: The agent followed the first part of instruction to *go down the hallway*. As the agent reached the end of the hallway, it was not able to find the second door to turn left. The agent then decided to roll back at step 4 with progress estimate decreased from 65% to 61%. The agent continued to go back towards the hallway but decided to roll back again at step 7. Although the agent was able to correct its errors made at the first few steps and *escape* from the hallway leading to the dead end, it ends up unsuccessful.

References

- [1] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. [2](#) [6](#)
- [2] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2018. [1](#) [2](#) [5](#) [6](#)
- [3] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991. [2](#)
- [4] M. Bhardwaj, S. Choudhury, and S. Scherer. Learning heuristic search via imitation. *arXiv preprint arXiv:1707.03034*, 2017. [3](#)
- [5] S. Brodeur, E. Perez, A. Anand, F. Golemo, L. Celotti, F. Strub, J. Rouat, H. Larochelle, and A. Courville. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*, 2017. [1](#)
- [6] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. [6](#)
- [7] A. Das, S. Datta, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. [1](#) [2](#)
- [8] A. Das, G. Gkioxari, S. Lee, D. Parikh, and D. Batra. Neural Modular Control for Embodied Question Answering. In *Proceedings of the Conference on Robot Learning (CoRL)*, 2018. [2](#)
- [9] H. de Vries, K. Shuster, D. Batra, D. Parikh, J. Weston, and D. Kiela. Talk the walk: Navigating new york city through grounded dialogue. *arXiv preprint arXiv:1807.03367*, 2018. [2](#)
- [10] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. [3](#)
- [11] R. E. Fikes, P. E. Hart, and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288, 1972. [2](#)
- [12] D. Fried, R. Hu, V. Cirik, A. Rohrbach, J. Andreas, L.-P. Morency, T. Berg-Kirkpatrick, K. Saenko, D. Klein, and T. Darrell. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems (NIPS)*, 2018. [1](#) [2](#) [3](#) [5](#) [6](#) [7](#) [9](#)
- [13] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. [1](#) [2](#)
- [14] C.-Y. Ma, J. Lu, Z. Wu, G. AlRegib, Z. Kira, R. Socher, and C. Xiong. Self-monitoring navigation agent via auxiliary progress estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019. [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#)
- [15] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. [2](#)
- [16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning (ICML)*, pages 1928–1937, 2016. [3](#)
- [17] A. Mousavian, A. Toshev, M. Fiser, J. Kosecka, and J. Davidson. Visual representations for semantic target driven navigation. *arXiv preprint arXiv:1805.06066*, 2018. [2](#)
- [18] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016. [3](#)
- [19] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017. [1](#)
- [20] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. [2](#) [6](#)
- [21] X. Wang, W. Xiong, H. Wang, and W. Y. Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *European Conference on Computer Vision (ECCV)*, 2018. [1](#) [2](#) [6](#)
- [22] G. Wayne, C.-C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, J. Rae, P. Mirowski, J. Z. Leibo, A. Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018. [2](#)
- [23] C. M. Wilt and W. Ruml. Building a heuristic for greedy search. In *Eighth Annual Symposium on Combinatorial Search*, 2015. [3](#)
- [24] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018. [1](#) [2](#)
- [25] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. [1](#)
- [26] Y. Xu, A. Fern, and S. W. Yoon. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, pages 2041–2046, 2007. [3](#)
- [27] H. Yu, X. Lian, H. Zhang, and W. Xu. Guided feature transformation (gft): A neural language grounding module for embodied agents. *arXiv preprint arXiv:1805.08329*, 2018. [2](#)