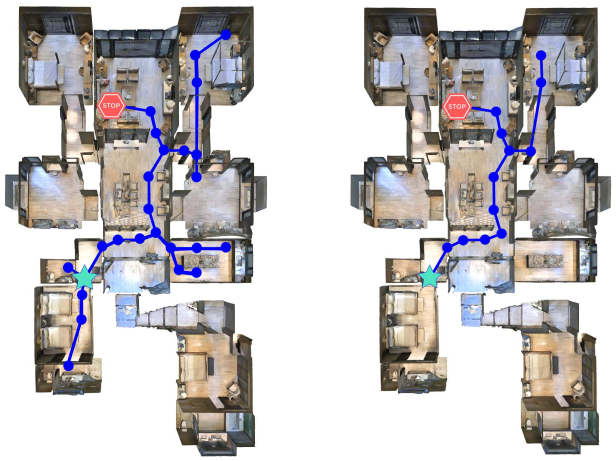
Tactical Rewind: Self-Correction via Backtracking in Vision-and-Language Navigation

Liyiming Ke[1]* Xiujun Li[1,2] Yonatan Bisk[1] Ari Holtzman[1] Zhe Gan[2]
Jingjing Liu[2] Jianfeng Gao[2] Yejin Choi[1,3] Siddhartha Srinivasa[1]
[1]Paul G. Allen School of Computer Science & Engineering, University of Washington
[2]Microsoft Research AI [3]Allen Institute for Artificial Intelligence
{kayke, xiujun, ybisk, ahai, yejin, siddh}@cs.washington.edu
{xiul, zhgan, jingjl, jfgao}@microsoft.com

Abstract

*We present the Frontier Aware Search with backTracking (*FAST*) Navigator, a general framework for action decoding, that achieves state-of-the-art results on the Room-to-Room (R2R) Vision-and-Language navigation challenge of Anderson et. al. (2018). Given a natural language instruction and photo-realistic image views of a previously unseen environment, the agent was tasked with navigating from source to target location as quickly as possible. While all current approaches make local action decisions or score entire trajectories using beam search, ours balances local and global signals when exploring an unobserved environment. Importantly, this lets us act greedily but use global signals to backtrack when necessary. Applying* FAST *framework to existing state-of-the-art models achieved a 17% relative gain, an absolute 6% gain on Success rate weighted by Path Length (SPL).*[1]



(a) SoTA Beam Search (b) FAST NAVIGATOR

Figure 1. Top-down view of the trajectory graphs for beam search and FAST. Blue Star is the start and Red Stop is the target.

1. Introduction

When reading an instruction (e.g. "*Exit the bathroom, take the second door on your right, pass the sofa and stop at the top of the stairs* ."), a person builds a mental map of how to arrive at a specific location. This map can include landmarks, such as the second door, and markers such as reaching the top of the stairs. Training an embodied agent to accomplish such a task with access to only ego-centric vision and individually supervised actions requires building rich multi-modal representations from limited data [2].

Most current approaches to Vision-and-Language Navigation (VLN) formulate the task to use the seq2seq (or encoder-decoder) framework [21], where language and vision are encoded as input and an optimal action sequence is decoded as output. Several subsequent architectures also use this framing; however, they augment it with important advances in attention mechanisms, global scoring, and beam search [2, 13, 10].

Inherent to the seq2seq formulation is the problem of *exposure bias* [19]: a model that has been trained to predict one-step into the future given the ground-truth sequence cannot perform accurately given its self-generated sequence. Previous work with seq2seq models attempted to address this using student forcing and beam search.

Student forcing exposes a model to its own generated sequence during training, teaching the agent how to recover. However, once the agent has deviated from the correct path, the original instruction no longer applies. The Supplementary Materials (§A.1) show that student forcing cannot solve the exposure bias problem, causing the confused agent to fall into loops.

*Work done partially as an intern at MSR
[1]The code is available at https://github.com/Kelym/FAST.

Beam search, at the other extreme, collects multiple global trajectories to score and incurs a cost proportional to the number of trajectories, which can be prohibitively high. This approach runs counter to the goal of building an agent that can efficiently navigate an environment: No one would likely deploy a household robot that re-navigates an entire house 100 times[2] before executing each command, even if it ultimately arrives at the correct location. The top performing systems on the VLN leaderboard[3] all require broad exploration that yields long trajectories, causing poor SPL performance (**S**uccess weighted by **P**ath **L**ength [1]).

To alleviate the issues of exposure bias and expensive, inefficient beam-search decoding, we propose the **F**rontier **A**ware **S**earch with back**T**racking(FAST NAVIGATOR). This framework lets agents *compare partial paths of different lengths* based on local and global information and then backtrack if it discerns a mistake. Figure 1 shows trajectory graphs created by the current published state-of-the-art (SoTA) agent using beam search versus our own.

Our method is a form of asynchronous search, which combines global and local knowledge to score and compare partial trajectories of different lengths. We evaluate our progress to the goal by modeling how closely our previous actions align with the given text instructions. To achieve this, we use a *fusion* function, which converts local action knowledge and history into an estimated score of progress. This score determines which local action to take and whether the agent should backtrack. This insight yields significant gains on evaluation metrics relative to existing models. The primary contributions of our work are:

- A method to alleviate the exposure bias of action decoding and expensiveness of beam search.
- An algorithm that makes use of asynchronous search with neural decoding.
- An extensible framework that can be applied to existing models to achieve significant gains on SPL.

2. Method

The VLN challenge requires an agent to carry out a natural language instruction in photo-realistic environments. The agent takes an input instruction $\mathcal{X}$, which contains several sentences describing a desired trajectory. At each step t, the agent observes its surroundings $\mathcal{V}_t$. Because the agent can look around for 360 degrees, $\mathcal{V}_t$ is in fact a set of $K = 36$ different views. We denote each view as $\mathcal{V}_t^k$. Using this multimodal input, the agent is trained to execute a sequence of actions $a_1, a_2, ..., a_T \in \mathcal{A}$ to reach a desired location. Consistent with recent work [13, 10], we use a panoramic action space, where each action corresponds to moving towards one of the K views, instead of R2R's original primitive action space (i.e, *left*, *right*, etc.) [2, 23]. In addition, this formulation includes a *stop* action to indicate that the agent has reached its goal.

[2]This is calculated based on the length of SPEAKER-FOLLOWER agent paths and human paths on the R2R dataset.

[3]https://evalai.cloudcv.org/web/challenges/challenge-page/97/leaderboard/270

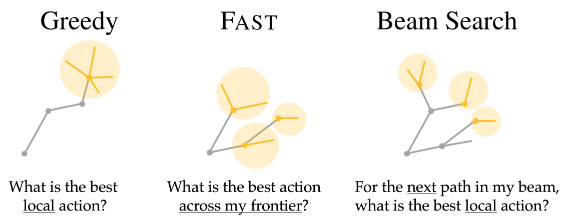

Greedy
FAST
Beam Search
What is the best local action?
What is the best action across my frontier?
For the next path in my beam, what is the best local action?

Figure 2. All VLN agents are performing a search. The orange areas highlight the frontier for different navigation methods.

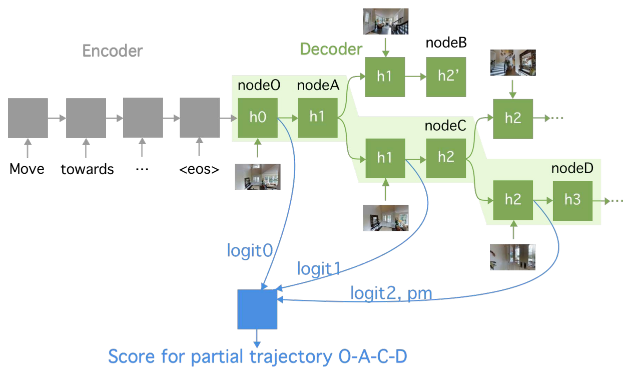
2.1. Learning Signals

Key to progress in visual navigation is that *all* VLN approaches performs a search (Figure 2). Current work often goes toward two extremes: using only local information, e.g. greedy decoding, or fully sweeping multiple paths simultaneously, e.g. beam search. To build an agent that can navigate an environment successfully and efficiently, we leverage both ***local*** and ***global*** information, letting the agent make a local decision while remaining aware of its global progress and efficiently backtracking when the agent discerns a mistake. Inspired by previous work [10, 13], our work uses three learning signals:

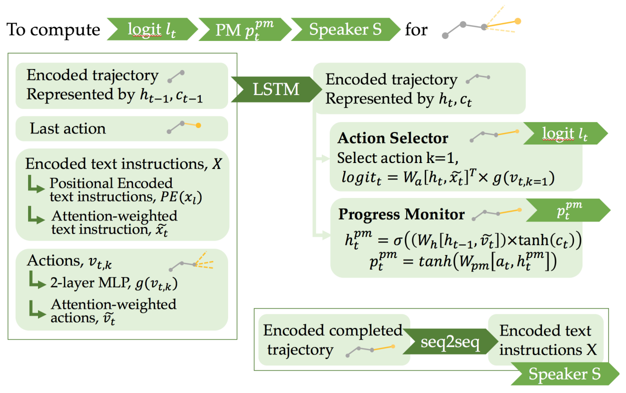
LOGIT l_t: local distribution over action. The logit of the action chosen at time t is denoted l_t. Specifically, the original language instruction is encoded via LSTM. Another LSTM acts as a decoder, using attention mechanism to generate logits over actions. At each time step t of decoding, logits are calculated by taking the dot product of the decoder's hidden state and each candidate action a_t^i.

PM p_t^{pm}: global progress monitor. It tracks how much of an instruction has been completed [13]. Formally, the model takes as input the (decoder) LSTM's current cell state, c_t, previous hidden state, h_{t-1}, visual inputs, $\mathcal{V}_t$, and attention over language embeddings, α_t to compute a score p_t^{pm}. The score ranges between [-1,1], indicating the agent's normalized progress. Training this indicator regularizes attention alignments, helping the model learn language-to-vision correspondences that it can use to compare multiple trajectories.

SPEAKER $\mathcal{S}$: global scoring. Given a sequence of visual observations and actions, we train a seq2seq captioning model as a "speaker" [10] to produce a textual description. Doing so provides two benefits: (1) the new speaker can automatically annotate new trajectories in the environment with the synthetic instructions, and (2) the speaker can score the likelihood that a given trajectory will correspond to the original instruction.



(a) Instructions and visual observations are encoded as hidden vectors defining multiple paths through the world. These vectors can then be accumulated to score a sequence of actions.


To compute
logit l_t
PM p_t^{pm}
Speaker S
for
Encoded trajectory
Represented by h_{t-1}, c_{t-1}
Last action
Encoded text instructions, X
Positional Encoded text instructions, $PE(x_l)$
Attention-weighted text instruction, $\tilde{x}_t$
Actions, $v_{t,k}$
2-layer MLP, $g(v_{t,k})$
Attention-weighted actions, $\tilde{v}_t$
LSTM
Encoded trajectory
Represented by h_t, c_t
Action Selector
logit l_t
Select action k=1,
$logit_t = W_a[h_t, \tilde{x}_t]^T \times g(v_{t,k=1})$
Progress Monitor
p_t^{pm}
$h_t^{pm} = \sigma((W_h[h_{t-1}, \tilde{v}_t]) \times \tanh(c_t))$
$p_t^{pm} = tanh(W_{pm}[a_t, h_t^{pm}])$
Encoded completed trajectory
seq2seq
Encoded text instructions X
Speaker S

(b) At each time step, the predicted action sequence and visual observation are fed into an attention module with the encoded instruction, to produce both the logits for the next actions and a progress monitor score.

Figure 3. (a). How the three signals are extracted from the partial trajectory in a seq2seq VLN framework; (b). How to compute the three signals.

2.2. Framework

We now introduce an extendible framework[4] that integrates the preceding three signals (l_t, p_t^{pm}, $\mathcal{S}$)[5] and to train new indicators, equipping an agent to answer:

1. Should we backtrack?
2. Where should we backtrack to?
3. Which visited node is most likely to be the goal?
4. When does it terminate this search?

These questions pertain to all existing approaches in navigation task. In particular, greedy approaches never backtrack and do not compare partial trajectories. Global beam search techniques always backtrack but can waste efforts. By taking a more principled approach to modeling navigation as graph traversal, our framework permits nuanced and adaptive answers to each of these questions.

For navigation, the graph is defined by a series of locations in the environment, called nodes. For each task, the agent is placed at a starting node, and the agent's movement in the house creates a trajectory comprised of a sequence of $<$*node* u, *action* a $>$ pairs. We denote a *partial* trajectory up to time t as τ_t, or the set of physical locations visited and the action taken at each point:

$$\tau_t = \{(u_i, a_i)\}_{i=1}^{t} \qquad (1)$$

For any partial trajectory, the last action is *proposed* and evaluated, but not *executed*. Instead, the model chooses whether to expand a partial trajectory or execute a *stop* action to complete the trajectory. Importantly, this means that every node the agent visited can serve as a possible final destination. The agent moves in the environment by choosing to extend a partial trajectory: it does this by moving to the last node of the partial trajectory and executing its last action to arrive at a new node. The agent then realizes the actions available at the new node and collects them to build a set of new partial trajectories.

At each time step, the agent must (1) access the set of partial trajectories it has not expanded, (2) access the completed trajectories that might constitute the candidate path, (3) calculate the accumulated cost of partial trajectories and the expected gain of its proposed action, and (4) compares all partial trajectories.

To do so, we maintain two priority queues: a *frontier queue*, $\mathcal{Q}_F$, for partial trajectories, and a *global candidate queue*, $\mathcal{Q}_C$, for completed trajectories. These queues are sorted by local $\mathcal{L}$ and global $\mathcal{G}$ scores, respectively. $\mathcal{L}$ scores the quality of all partial trajectories with their proposed actions and maintains their order in $\mathcal{Q}_F$; $\mathcal{G}$ scores the quality of completed trajectories and maintains the order in $\mathcal{Q}_C$.

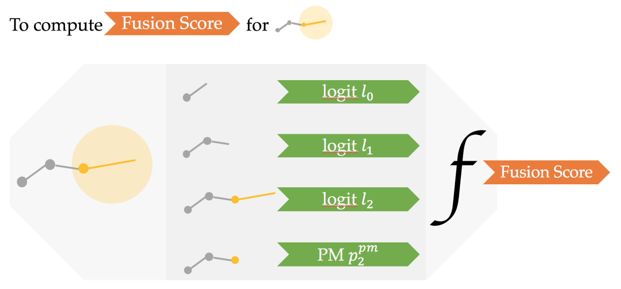
In §4.3, we explore alternative formulas for $\mathcal{L}$ and $\mathcal{G}$. For example, we define $\mathcal{L}$ and $\mathcal{G}$ using the signals described in §2.1 and a function, f, that is implemented as a neural network.

$$\mathcal{L} \quad \leftarrow \Sigma_{0\to t}\, l_i \qquad (2)$$
$$\mathcal{G} \quad \leftarrow f(\mathcal{S}, p_t^{pm}, \Sigma_{0\to t}\, l_i, \ldots) \qquad (3)$$

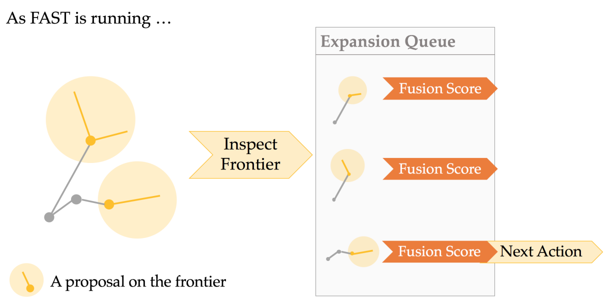
To allow the agent to efficiently navigate and follow the instruction, we use an approximation of the D* search. FAST expands its optimal partial trajectory until it decides to backtrack (**Q1**). It decides on where to backtrack (**Q2**) by ranking all partial trajectories. To propose the final goal location (**Q3 & Q4**), the agent ranks the completed global trajectories in candidate queue $\mathcal{Q}_C$. We explore these questions in more detail below.

[4]Figure 3(a) shows an example of integrating the three signals in a seq2seq framework.

[5]Figure 3(b) shows how to compute the three signals.



(a) Both local $\mathcal{L}$ and global $\mathcal{G}$ scores can be trained to condition on arbitrary information. Here, we show the fusion of historical logits and progress monitor information into a single score.



(b) An expansion queue maintains all possible next actions from all partial trajectories. The options are sorted by their scores (Figure 4(a)) in order to select the next action.

Figure 4. Arbitrary signals can be computed from partial trajectories to learn a scoring function (left) that ranks all possible actions in our expansion queue (right). This provides a flexible and extendible framework for optimal action decoding.

Q1: Should we backtrack? When an agent makes a mistake or gets lost, backtracking lets it move to a more promising partial trajectory; however, retracing steps increases the length of the final path. To determine when it is worth incurring this cost, we proposed two simple strategies: *explore* and *exploit*.

1. *Explore* always backtracks to the most promising partial trajectory. This approach resembles beam search, but, rather than simply moving to the next partial trajectory in the beam, the agent computes the most promising node to backtrack to (**Q2**).
2. *Exploit*, in contrast, commits to the current partial trajectory, always executing the best action available at the agent's current location. This approach resembles greedy decoding, except that the agent backtracks when it is confused (i.e, when the best local action causes the agent to revisit a node, creating a loop; see the SMNA examples in Supplementary Materials §A.1).

Q2: Where should we backtrack to? Making this decision involves using $\mathcal{L}$ to score all partial trajectories. Intuitively, the better a partial trajectory aligns with a given description, the higher the value of $\mathcal{L}$. Thus, if we can assume the veracity of $\mathcal{L}$, the agent simply returns to the highest scoring node when backtracking. Throughout this paper, we explore several functions for computing $\mathcal{L}$, but we present two simple techniques here, each acting over the sequence of actions that comprise a trajectory:

1. *Sum-of-log* $\sum_{0\to t} \log p_i$ sums the log-probabilities of every previous action, thereby computing the probability of a partial trajectory.
2. *Sum-of-logits* $\sum_{0\to t} l_i$ sums the unnormalized logits of previous actions, which outperforms summing probabilities. These values are computed using an attention mechanism over the hidden state, observations, and language. In this way, their magnitude captures how well the action was aligned with the target description (this information is lost during normalization).[6]

Finally, during exploration, the agent implicitly constructs a "mental map" of the visited space. This lets it search more efficient by refusing to revisit nodes, unless they lead to a high-value unexplored path.

Q3: Which visited node is most likely to be the goal? Unlike existing approaches, FAST considers every point that the agent has visited as a candidate for the final destination,[7] meaning we must rerank all candidates. We achieve this using $\mathcal{G}$, a trainable neural network function that incorporates all global information for each candidate and ranks them accordingly. Figure 4(a) shows a simple visualization.

We experimented with several approaches to compute $\mathcal{G}$, e.g., by integrating $\mathcal{L}$, the progress monitor, speaker score, and a trainable ensemble in (§4.3).

Q4: When do we terminate the search? The flexibility of FAST allows it to recover both the greedy decoding and beam search framework. In addition, we define two alternative stopping criteria:

1. When a partial trajectory decides to terminate.
2. When we have expanded M nodes. In §3 we ablate the effect of choosing a different M.

2.3. Algorithm

We present the algorithm flow of our FAST framework. When an agent is initialized and placed on the starting node, both the candidate and frontier queues are empty. The agent

[6]This is particularly problematic when an agent is lost. Normalizing many low-value logits can yield a comparatively high probability (e.g. uniform or random). We also experiment with variations of this approach (e.g. means instead of sums) in §4.

[7]There can be more than one trajectory connecting the starting node to each visited node.

Algorithm 1 FAST NAVIGATOR

```
1: procedure FAST NAVIGATOR
2:     $Q_F^{sort=\mathcal{L}}, Q_C^{sort=\mathcal{G}} = \{\}, \{\}$
3:     $Q_F \leftarrow (u_0, a_0 = \texttt{None})$                ▷ Initial Proposal
4:     $\hat{\tau} \leftarrow \varnothing$
5:     $M \leftarrow \varnothing$                                  ▷ Mental Map
6:     while $Q_F \neq \varnothing$ and *stop criterion* do
7:         if *need backtrack* or $\hat{\tau} == \varnothing$ then
8:             $\hat{\tau} \leftarrow Q_F$.pop
9:         end if
10:        $\hat{u}_{t-1}, \hat{a}_{t-1} \leftarrow \hat{\tau}$.last
11:        if $(\hat{u}_{t-1}, \hat{a}_{t-1}) \in M$ then
12:            $u_t \leftarrow M(\hat{u}_{t-1}, \hat{a}_{t-1})$
13:        else
14:            $u_t \leftarrow$ move to $u_{t-1}$ and execute $a_{t-1}$
15:            $M(\hat{u}_{t-1}, \hat{a}_{t-1}) \leftarrow u_t$
16:        end if
17:        for $a_k$ in best $K$ next actions do
18:            $Q_F \leftarrow Q_F \cup \{\hat{\tau} + (u_t, a_k)\}$
19:        end for
20:        $Q_C \leftarrow Q_C \cup \hat{\tau}$
21:        $\hat{\tau} \leftarrow \hat{\tau} + (u_t, a^*)$ where $a^*$ is the best action
22:    end while
23:    return $Q_C$.pop
24: end procedure
```

then adds all possible next actions to the frontier queue and adds its current location to the candidate queue:

$$Q_F \leftarrow Q_F + \forall_{i\in K}\{\tau_0 \cup (u_0, a_i)\} \quad (4)$$
$$Q_C \leftarrow Q_C + \tau_0 \quad (5)$$

Now that the Q_F is not empty and the stop criterion is not met, FAST can choose the best partial trajectory from the frontier queue under the local scoring function:

$$\hat{\tau} \leftarrow \arg\max_{\tau_i} \mathcal{L}(Q_F) \quad (6)$$

Following $\hat{\tau}$, we perform the final action proposal, a_t, to move to a new node (location in the house). FAST can now update the candidate queue with this location and the frontier queue with all possible new actions. We then either continue, by exploiting the available actions at the new location, or backtrack, depending on the choice of backtrack criteria. We repeat this process until the model chooses to stop and returns the best candidate trajectory.

$$\tau^* \leftarrow \arg\max_{\tau} \mathcal{G}(Q_C) \quad (7)$$

Algorithm 1 more precisely outlines the full procedure for our approach. §4.3 details the different approaches to scoring partial and complete trajectories.

3. Experiments

We evaluate our approach using the Room-to-Room (R2R) dataset [2]. At the beginning of the task, the agent receives a natural language instruction and a specific start location in the environment; the agent must navigate to the target location specified in the instruction as quickly as possible. R2R is built upon the Matterport3D dataset [5], which consists of >194K images, yielding 10,800 panoramic views ("nodes") and 7,189 paths. Each path is matched with three natural language instructions.

3.1. Evaluation Criteria

We evaluate our approach on the following metrics in the R2R dataset:

TL **Trajectory Length** measures the average length of the navigation trajectory.

NE **Navigation Error** is the mean of the shortest path distance in meters between the agent's final location and the goal location.

SR **Success Rate** is the percentage of the agent's final location that is less than 3 meters away from the goal location.

SPL **Success weighted by Path Length [1]** trades-off SR against TL. Higher score represents more efficiency in navigation.

3.2. Baselines

We compare our results to four published baselines for this task.[8]

- RANDOM: an agent that randomly selects a direction and moves five step in that direction [2].
- SEQ2SEQ: the best performing model in the R2R dataset paper [2].
- SPEAKER-FOLLOWER [10]: an agent trained with data augmentation from a speaker model on the panoramic action space.
- SMNA [13]: an agent trained with a visual-textual co-grounding module and a progress monitor on the panoramic action space.[9]

3.3. Our Model

As our framework provides a flexible design space, we report performance for two versions:

- FAST(short) uses the exploit strategy. We use the sum of logits *fusion* method to compute $\mathcal{L}$ and terminate when the best local action is stop.

[8]Some baselines on the leader-board are not yet public when submitted; therefore, we cannot compare with them directly on the training and validation sets.

[9]Our SMNA implementation matches published validation numbers. All our experiments are based on full re-implementations.

	Model	Validation Seen TL	NE	SR	SPL	Validation Unseen TL	NE	SR	SPL	Test Unseen TL	NE	SR	SPL
	RANDOM	9.58	9.45	0.16	-	9.77	9.23	0.16	-	9.93	9.77	0.13	0.12
	Seq2seq	11.33	6.01	0.39	-	8.39	7.81	0.22	-	8.13	7.85	0.20	0.18
	Our baseline SMNA	11.69	3.31	0.69	0.63	12.61	5.48	0.47	0.41	-	-	-	-
Greedy	SMNA	-	-	-	-	-	-	-	-	18.04	5.67	0.48	0.35
	SPEAKER-FOLLOWER	-	-	-	-	-	-	-	-	**14.82**	6.62	0.35	0.28
	+ FAST (short)					21.17	4.97	0.56	0.43	22.08	**5.14**	**0.54**	**0.41**
Beam	SMNA	-	3.23	0.70	-	-	5.04	0.57	-	373.09	4.48	**0.61**	0.02
	SPEAKER-FOLLOWER	-	3.88	0.63	-	-	5.24	0.50	-	1,257.30	4.87	0.53	0.01
	+ FAST (long)	188.06	3.13	0.70	0.04	224.42	4.03	**0.63**	0.02	**196.53**	**4.29**	**0.61**	**0.03**
	Human	-	-	-	-	-	-	-	-	11.85	1.61	0.86	0.76

Table 1. Our results and SMNA re-implementation are shown in gray highlighted rows. **Bolding** indicates the best value per section and **blue** indicates best values overall. We include both a short and long version of our approach to compare to existing models greedy and beam search approaches.

- FAST(long) uses the explore strategy. We again use the sum of logits for fusion, terminating the search after fixed number of nodes and using a trained neural network reranker to select the goal state $\mathcal{G}$.

3.4. Results

Table 1 compares the performance of our model against published numbers of existing models. Our approach significantly outperforms the existing model in terms of efficiency, matching the best overall success rate despite taking 150 - 1,000 fewer steps. This efficiency gain can be seen in the SPL metric, where our models outperform previous approaches in every setting. Note that our short trajectory model appreciably outperforms current approaches in both SR and SPL. If our agent could continue exploring, it matches existing peak success rates in half of the steps (196 vs 373).

Validation Unseen	SR (%)	SPL (%)	TL
SPEAKER-FOLLOWER	37	28	15.32
+ FAST	43 (**+6**)	29 (**+1**)	20.63
SMNA	47	41	12.61
+ FAST	56 (**+9**)	43 (**+2**)	21.17

Table 2. Plug-n-play performance gains achieved by adding FAST to current SoTA models.

Another key advantage of our technique is how simple it is to integrate with current approaches to achieve dramatic performance gains. Table 2 shows how the sum-of-logits fusion method enhances the two previously best performing models. Simply changing their greedy decoders to FAST with no added global information and therefore no reranking yields immediate gains of 6 and 9 points in success rate for SPEAKER-FOLLOWER and SMNA, respectively. Due to those models' new ability to backtrack, the trajectory lengths increase slightly. However, the success rate increases so much that SPL increases, as well.

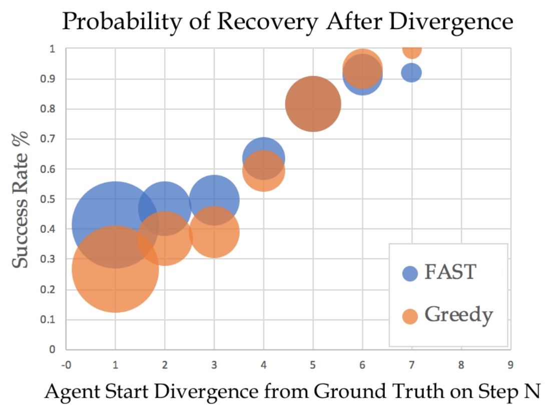


Figure 5. Circle sizes represent the what percentage of agents diverge on step N. Most divergences occur in the early steps. FAST recovers from early divergences.

4. Analysis

Here, we isolate the effects of local and global knowledge, the importance of backtracking, and various stopping criteria. In addition, we include three qualitative intuitive examples to illustrate the model's behavior in the Supplementary Materials (§A.1). We can perform this analysis because our approach has access to the same information as previous architectures, but it is more efficient. Our claims and results are general, and our FAST approach should benefit future VLN architectures.

4.1. Fixing Your Mistakes

To investigate the degree to which models benefit from backtracking, Figure 5 plots a model's likelihood of successfully completing the task after making its first mistake at each step. We use SMNA as our greedy baseline. Our analysis finds that the previous SoTA model makes a mistake at the very first action 40% of the time. Figure 5 shows the effect of this error: the greedy approach, if made a mistake at its first step, has a <30% chance of successfully completing the task. In contrast, because FAST detects its mistake, it returns to the starting position and tries again. This simple one-step backtracking increases its likelihood of success by over 10%. In fact, the greedy approach is equally successful only if it progresses over halfway through the instruction without making a mistake.

4.2. Knowing When To Stop Exploring

The stopping criterion balances exploration and exploitation. Unlike previous approaches, our framework lets us compare different criteria and offers the flexibility to determine which is optimal for a given domain. The best available stopping criterion for VLN is not necessarily the best in general. We investigated the number of nodes to expand before terminating the algorithm, and we plot the resulting success rate and SPL in Figure 6. One important finding is that the model's success rate, though increasing with more nodes expanded, does not match the oracle's rate, i.e., as the agent expands 40 nodes, it has visited the true target node over 90% of the time but cannot recognize it as the final destination. This motivates an analysis of the utility of our global information and whether it is truly predictive (Table 4), which we investigate further in §4.3.

4.3. Local and Global Scoring

As noted in §2.3, core to our approach are two queues, frontier queue for expansion and the candidate queue for proposing the final candidate. Each queue can use arbitrary information for scoring (partial) trajectories. We now compare the effects of combining different set of signals for scoring each queue.

Fusion methods for scoring partial trajectories An ideal model would include as much global information as possible when scoring partial trajectories in the frontier expansion queue. Thus, we investigated several sources of pseudo-global information and ten different ways to combine them. The first four use only local information, while the others attempts to fuse local and global information.

The top half of Table 3 shows the performance when considering only local information providers. For example, the third row of the table shows that summing the logit scores of nodes along the partial trajectory as the $\mathcal{L}$ score for that trajectory achieves an SR score of 56.66. Note although all information originates with the same hidden vectors, the values computed and how they are aggregated substantially affect performance. Overall, we find that summing unnormalized logits (the 3rd row) performs the best considering its outstanding SR. This suggests that important activation information in the network outputs is being thrown away by normalization and therefore discarded by other techniques.

The bottom part of Table 3 explores ways of combining local and global information providers. These are motivated by beam-rescoring techniques in previous work (e.g., multiplying by the normalized progress monitor score). Correctly integrating signals is challenging, in part due to differences in scale. For example, the logit is unbounded (+/-), log probabilities are unbounded in the negative, and the progress monitor is normalized to a score between 0 and 1. Unfortunately, direct integration of the progress monitor did not yield promising results, but future signals may prove more powerful.

Fusion methods for ranking complete trajectories . Previous work [10] used state-factored beam search to

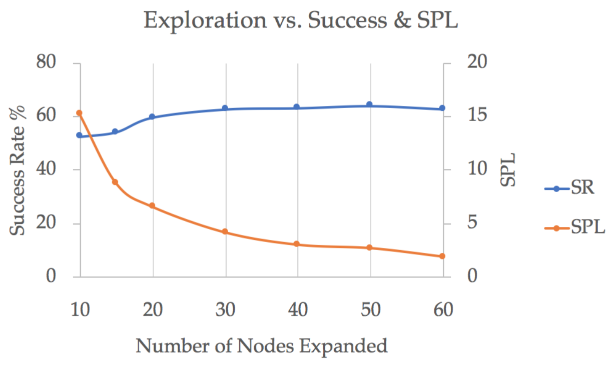


Figure 6. The SR increases with the number of nodes explored before plateauing, while SPL (which is extremely sensitive to length) continually decreases with added exploration.

Heur/step	*Combine*	*SR*	*SPL*	*Len*
logit	mean	53.89	**44.74**	14.80
log prob	mean	53.85	44.14	15.57
logit	sum	**56.66**	43.64	21.17
log prob	sum	56.23	42.66	21.70
logit	mean / pm	53.00	44.51	13.67
log prob	mean / pm	53.72	44.64	13.85
logit	mean * pm	54.78	**44.70**	15.91
log prob	mean * pm	55.04	43.70	17.45
logit	sum * pm	50.95	41.28	20.25
log prob	sum * pm	**56.15**	43.19	21.55

Table 3. Performance of different fusion methods for scoring partial trajectories. Tested on the validation unseen set.

generate M candidates and rank the complete trajectories using probability of speaker and follower scores $\text{argmax}_{r\in R(d)} P_S(d|r)^{\lambda} * P_F(d|r)^{(1-\lambda)}$. In addition to the speaker and progress monitor scores used by previous models, we also experiment with using $\mathcal{L}$ to compute $\mathcal{G}$. To inspect the performance of using different fusion methods, we ran FAST NAVIGATOR to expand 40 nodes on the frontier and collect candidate trajectories. Table 4 shows the performance of different fusion scores that rank complete trajectories. We see that most techniques have a limited understanding of the global task's goal and formulation. We do, however, find a significant improvement on unseen trajectories when all signals are combined. For this we train a multi-layer perceptron to aggregate and weight our predictors. Note that any improvements to the underlying models or new features introduced by future work will directly correlate to gains in this component of the pipeline.

The top line of Table 4, shows oracle's performance. This indicates how far current global information providers have yet to achieve. Closing this gap is an important direction for future work.

	Train	Val Seen	Val Unseen
Oracle	99.13	92.85	90.20
$\Sigma\ l_i$	78.78	62.49	56.49
$\mu\ l_i$	85.78	66.99	54.41
$\Sigma\ p_i$	91.25	68.56	56.15
$\mu\ p_i$	91.60	69.34	58.75
p_t^{pm}	66.71	53.67	50.15
$\mathcal{S}$	69.99	53.77	43.68
All	90.16	**71.00**	**64.03**

Table 4. Success rate using seven different fusion scores as $\mathcal{G}$ to rerank the destination node from the candidate pool.

4.4. Intuitive Behavior

The Supplementary Materials (§A.1) provide three real examples to show how our model performs when compared to greedy decoding (SMNA model). It highlights how the same observations can lead to drastically different behaviors during an agent's rollout. Specifically, in Figures A1 and A2, the greedy decoder is forced into a behavioral loop because only local improvements are considered. Using FAST clearly shows that even a single backtracking step can free the agent of poor behavioral choices.

5. Related Work

Our work focuses on and complements recent advances in Vision-and-Language Navigation (VLN) as introduced by [2], but many aspects of the task and core technologies date back much further. The natural language community has explored instruction following using 2D maps [17, 14] and computer-rendered 3D environments [16]. Due to the enormous visual complexity of real-world scenes, the VLN literature usually builds on computer vision work from referring expressions [15, 24], visual question answering [3], and ego-centric QA that requires navigation to answer questions [11, 8, 9]. Finally, core to the our work is the field of search algorithm, dating back to the earliest days of AI [18, 20], but largely absent from recent VLN literature that tends to focuses more on neural architecture design.

During publishing the Room-to-Room dataset (VLN), [2] introduced the "student forcing" method for seq2seq model. Later work integrated a planning module to combined model-based and model-free reinforcement learning to better generalize to unseen environments [23], and a Cross-Modal Matching method that enforces cross-modal grounding both locally and globally via reinforcement learning [22]. Two substantial improvements came from panoramic action spaces and a "speaker" model trained to enable data augmentation and trajectory reranking for beam search [10]. Most recently, [13] leverages a visual-textual co-grounding attention mechanism to better align the instruction and visual scenes and incorporates a progress monitor to estimate the agent's current progress towards a goal. These approaches require beam search for peak SR. Beam search techniques can unfortunately lead to long trajectories when exploring unknown environments. This limitation motivates the work we present here. Existing approaches trade off a high success rate and long trajectories: greedy decoding provides short, often incorrect paths, the beam search yields high success rates but long trajectories.

6. Conclusion

We present FAST NAVIGATOR, a framework for using asynchronous search to boost any VLN navigator by enabling explicit backtrack when an agent detects if it is lost. This framework can be easily plugged into the most advanced agents to immediately improve their efficiency. Further, empirical results on the Room-to-Room dataset show that our agent achieves state-of-the-art Success Rates and SPLs. Our search-based method is easily extendible to more challenging settings, e.g., when an agent is given a goal without any route instruction [6, 12], or a complicated real visual environment [7].

Acknowledgments

Partial funding provided by DARPA's CwC program through ARO (W911NF-15-1-0543), NSF (IIS-1524371, 1703166), National Institute of Health (R01EB019335), National Science Foundation CPS (1544797), National Science Foundation NRI (1637748), the Office of Naval Research, the RCTA, Amazon, and Honda.

References

[1] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 2, 5

[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, 8

[3] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (CVPR)*, pages 2425–2433, 2015. 8

[4] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96. ACM, 2005. 10

[5] 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. 5

[6] D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 8

[7] H. Chen, A. Shur, D. Misra, N. Snavely, and Y. Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 8

[8] 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)*, volume 5, page 6, 2018. 8

[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. 8

[10] 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 *Neural Information Processing Systems (NeurIPS)*, 2018. 1, 2, 5, 7, 8

[11] D. Gordon, A. Kembhavi, M. Rastegari, J. Redmon, D. Fox, and A. Farhadi. IQA: Visual question answering in interactive environments. In *Computer Vision and Pattern Recognition (CVPR)*, volume 1, 2018. 8

[12] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. M. Czarnecki, M. Jaderberg, D. Teplyashin, et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017. 8

[13] 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 *International Conference on Learning Representations (ICLR)*, 2019. 1, 2, 5, 8, 11

[14] H. Mei, M. Bansal, and M. R. Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI*, volume 1, page 2, 2016. 8

[15] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations (ICLR)*, 2017. 8

[16] D. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018. 8

[17] D. Misra, J. Langford, and Y. Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2017. 8

[18] J. Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984. 8

[19] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016. 1

[20] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016. 8

[21] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014. 1

[22] 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 *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 8

[23] 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. 2, 8

[24] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364. IEEE, 2017. 8