NavQ: Learning a Q-Model for Foresighted Vision-and-Language Navigation

Supplementary Material

6. Details on the Model and Training

6.1. Preliminaries on DUET

The baseline model for our NavQ agent, DUET [14], proposes a dual-scale action prediction strategy on a topological graph for the VLN task. Due to its generality, DUET's architecture has been adopted by many subsequent studies [15, 64, 71, 79, 87, 123, 141]. At each navigation timestep, it involves the following computation procedures:

(1) Input processing. The agent perceives the surrounding environment at its current location through a panoramic RGB observation. The panoramic image is discretized into $N = 36$ views (3 elevation angles times 12 heading angles) and processed by a frozen visual encoder. The agent also gets the feature for M^t visible objects (pre-defined or detected by an off-the-shelf detector, see Section 7.1 for details). The feature vectors for views and objects are concatenated and sent to a learnable panorama encoding module, which is implemented as a 2-layer Transformer. The results are $\{r_i^t\}_{i=1}^N$ and $\{o_i^t\}_{i=1}^{M^t}$ mentioned in Eq (1) and (3). On the other hand, the word embeddings of the instruction text is sent to another 9-layer Transformer to obtain the textual feature w.

(2) Graph update. The agent builds a topological graph G^t on the fly. The graph starts as a single node representing the starting position of the episode. VLN's task setting [3] assumes that the agent has access to the locations of navigable viewpoints around its current place. Thus, it can continuously expand its graph by incorporating neighboring nodes. There are two types of nodes on the graph: visited nodes and observed but unvisited nodes. For each visited node $\mathcal{N}$, the agent stores the mean of its view features ($\frac{1}{N}\sum_{i=1}^N r_i^t$, t is the step that it visits $\mathcal{N}$) as its feature. For each unvisited node $\mathcal{N}$, the agent maintains a list. If $\mathcal{N}$ is a neighboring node of the agent's location at step t, it identifies one of the N views that is closest to the direction of $\mathcal{N}$ and inserts its feature r_i^t into the list. (Note that an unvisited node can be observed by multiple visited nodes.) The agent takes the mean of this list as the feature for the corresponding node.

(3) Global action prediction. DUET features a dual-scale planning process. The coarse-scale branch outputs a probability across all the unvisited nodes on G^t (*i.e.*, the global candidates). It is based on a 4-layer Graph Transformer named as global encoder (GE). Each layer performs sequentially a cross-modal attention that interacts the textual feature w with the node features $\{v_i^t\}_{i=0}^{|G^t|}$, and a graph-aware self-attention that takes into account the structure of the graph to further process the node features. v_0^t is a zero vector representing a pseudo "stop" node. The outputs of GE are the updated node features $\{\hat{v}_i^t\}_{i=0}^{|G^t|}$, as in Eq (2). They are transformed into logits $\{s_i^{c,t}\}_{i=0}^{|G^t|}$ by an MLP.

(4) Local action and object prediction. The fine-scale branch of DUET outputs a probability across the unvisited neighbors of the current node (*i.e.*, the local candidates). It is based on a 4-layer Transformer named as local encoder (LE). Each layer performs sequentially a cross-modal attention that interacts the textual feature w with the concatenation of view features $\{r_i^t\}_{i=0}^N$ and object features $\{o_i^t\}_{i=1}^{M^t}$, and a self-attention that further processes the concatenation. r_0^t is a zero vector representing the "stop" action. The outputs of LE are the updated view features $\{\hat{r}_i^t\}_{i=0}^N$ and object features $\{\hat{o}_i^t\}_{i=1}^{M^t}$, as in Eq (3). They are transformed into action logits $\left\{s_i^{f,t}\right\}_{i=0}^N$ and object logits $\{s_i^{o,t}\}_{i=1}^{M^t}$ by two separate MLPs.

(5) Dynamic Fusion. DUET dynamically fuses the global prediction and local prediction to get the final action logits $\{s_i^t\}_{i=0}^{|G^t|}$. The fusing weight is obtained by sending the concatenation of v_0^t and r_0^t to an MLP and a Sigmoid funtion.

(6) Action execution. The agent selects a candidate node based on the fused probability. It then finds the shortest path from its current location to this node on G^t, and traverses along it. When the agent decides to "stop", it selects an object as its prediction according to the local object probability.

6.2. Details of the Q-Model

In this subsection we describe the training of the Q-model in more detail. To get each training sample, we randomly select a training scene and a starting node. Then, a partial trajectory is obtained by uniformly choosing an unvisited local candidate for a random number of steps. Based on this trajectory, the model input is formed as follows.

- For each node in the trajectory, we encode its 36 view images into visual features, and pool them into 12 vectors corresponding to the 12 heading directions. We take the natural language description of each view provided by LangNav [92] (extracted using BLIP [60]), encode them into textual features, and pool them into 12 vectors as well. The visual features and textual features are processed by linear projections and added together, forming the full node feature of shape $12 \times D$.
- For each action in the trajectory, we encode its orientation using sin and cos functions. The resulting vector is

linearly projected to D channels. For each local candidate actions at the current node (*i.e.*, the last node of the partial trajectory), we encode it in the same way to a D-dimensional vector.

- We arrange the node features and action features alternately following the order in the trajectory, and append the candidate features at the end. As illustrated in Figure 3, the input to the Q-model is a sequence of $13|\mathbf{T}|-1+C$ tokens, where $|\mathbf{T}|$ is the length (number of nodes) of the partial trajectory, while C is the number of local candidates. Each token is a D-dimensional vector.

The Q-model is implemented as a 4-layer Transformer. Apart from the traditional positional encoding that captures the order of tokens, we introduce an additional positional encoding to represent the token order within a node. Specifically, this encoding consists of 13 learnable tokens, which are added to the 13 input tokens corresponding to each node-action pair. Notably, the last positional token is added to each candidate token.

We adopt the method described in Section 3.3.1 to form the ground-truth Q-feature. Before delving into the implementation details, we first provide a more precise formulation of the rollout policy π used in our method. Given a partial trajectory $\mathbf{T}$ and a candidate action a (leading to node $\mathcal{A}$), $\mathbf{T}\cup\{\mathcal{A}\}$ is expanded to a full trajectory $\tilde{\mathbf{T}}$ under π. At each step, the agent randomly selects a feasible local candidate according to a uniform distribution, where feasibility means that this candidate node $\mathcal{N}$ ensures that the one-step-longer rollout path is the shortest path from $\mathbf{T}[-1]$ to $\mathcal{N}$. The agent terminates when there is no feasible candidate to choose. This formulation is consistent with the definition of the set of possible rollout trajectories, $\mathbb{T}$. In Section 3.3.1, we put forward a claim that for a given pair of partial trajectory $\mathbf{T}$ and node $\mathcal{N}$, there is at most one pair of (a,t) that makes $\mathrm{P}_\pi(\mathcal{N},t|\mathbf{T},a)>0$ under the policy π. This can be easily proved by contradiction. Suppose $\mathrm{P}_\pi(\mathcal{N},t_1|\mathbf{T},a_1)>0$ and $\mathrm{P}_\pi(\mathcal{N},t_2|\mathbf{T},a_2)>0$. If $t_1\neq t_2$, then there are two paths of different length going from $\mathbf{T}[-1]$ to $\mathcal{N}$. They cannot simultaneously be the shortest path and then cannot both be obtained under policy π. If $a_1\neq a_2$, then there are two different paths going from $\mathbf{T}[-1]$ to $\mathcal{N}$, containing $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Still, they cannot both be obtained under policy π. Therefore, the claim is proved, and we can use $t(\mathcal{N})$ to denote the unique rollout step t for each future node $\mathcal{N}$.

We now provide a practical implementation for computing $\boldsymbol{Q}(\mathbf{T},a)$. We first identify all the nodes $\mathcal{N}$ in the scene that satisfy the following condition: the shortest path from $\mathcal{N}$ to $\mathbf{T}[-1]$ passes through $\mathcal{A}$. We also record the rollout step $t(\mathcal{N})$ for each node as the hop of the shortest path from $\mathbf{T}[-1]$ to $\mathcal{N}$. We sort these nodes in ascending order based on the values of t and sequentially compute their rollout probabilities $\mathrm{P}_\pi(\mathcal{N},t(\mathcal{N})|\mathbf{T},a)$. Finally, we use Eq (8) to obtain $\boldsymbol{Q}(\mathbf{T},a)=\sum_{\mathcal{N}}\mathrm{P}_\pi(\mathcal{N},t(\mathcal{N})|\mathbf{T},a)\gamma^{t(\mathcal{N})}\boldsymbol{R}(\mathcal{N})$. As stated in Section 3.3.2, $\boldsymbol{R}(\mathcal{N})$ is an abstracted text-based feature. We set it to the average textual feature of the 36 views' natural language descriptions. The resulting $\boldsymbol{Q}(\mathbf{T},a)$ serves as the ground-truth Q-feature for candidate action a.

The Q-model is trained on the training split of MatterPort3D [3, 7], which is also shared by REVERIE [96] and SOON [143]'s training set. For experiments with additional scenes, we employ the scenes, graphs, and images generated by ScaleVLN [123], which consists of 800 scenes from HM3D [102] and 491 scenes from Gibson [128]. We do not use the trajectory annoations generated by ScaleVLN. For validation, we evaluate the Q-model on the val-unseen split of REVERIE.

6.3. Details of the Future Encoder

The proposed future encoder has the same Graph Transformer architecture as DUET's global encoder, but takes different input. We build an additional graph $\tilde{G}^t$ that shares topology with DUET's navigation graph G^t. For each unvisited node, we maintain a similar list as described in the step (2) of Section 6.1, while the contents of it are the Q-features related to the node instead of the view features. For each visited node, we extract the average feature for textual descriptions (*i.e.*, $\boldsymbol{R}(\mathcal{N})$) as the node feature.

The output of GE, FE, and LE are fused together by weighted addition. Thus, the Sigmoid function employed by DUET's dynamic fusion (Section 6.1, step (5)) is replaced by a Softmax function.

6.4. Training Tasks

The training of DUET consists of two stages: offline pre-training and online finetuning. In the pre-training stage, a batch of partial trajectories are sent to the model, which is trained to perform one of the following training tasks:

- MLM (masked language modeling). A random mask is applied to the instruction text, and the agent is asked to reconstruct the masked tokens. For this task, the cross-modal layers in GE/FE/LE use the node/view features as key and value, while the textual features are used as query. The output of them are summed together and processed by an MLP head for word prediction.
- SAP (single-step action prediction). The agent is asked to choose the best next-step action (among the global candidates) given a partial trajectory. The output action logits are supervised by cross-entropy loss, and the ground-truth is the candidate with the shortest distance to the destination. This loss is computed on the global, local, and fused logits in DUET. We further apply it to the future logits output by FE.
- OG (object grounding). The agent is asked to predict the correct object given a trajectory ending at a correct lo-

Table 5. Distribution of parameters in the NavQ agent. The listed modules from left to right are the panoramic encoder (Section 6.1, step (1)), the textual encoder (Section 6.1, step (1)), the global encoder (Section 6.1, step (3)), the future encoder (Section 6.3), the local encoder (Section 6.1, step (4)), the Q-model (Section 6.2), and the prediction heads for generating and fusing logits.

PE	TE	GE	FE	LE	QM	Heads	Total
15.2	87.6	37.9	39.2	37.8	30.5	4.1	252.4M

cation. The output object logits are supervised by cross-entropy loss.

- MRC (masked region classification). Similar to MLM, some of the input views and objects are masked, and the agent is asked to predict their semantic class. An MLP is appended after LE for prediction. The ground-truth semantic labels are the output class probability of a frozen classification model and a frozen detection model.

Our training stage 2 (Section 3.5) inherits the design of these tasks. The proposed progress-related sub-tasks are integrated into SAP. To be specific, we compute the ground-truth historical progress s_1 and distance to go s_2 for each global candidate (Section 3.4). We then clip them to $[0, 1]$, and discretize them into 5 bins. The output node features of GE and FE are sent to two separate MLPs to perform a 5-category classification task. The two cross-entropy losses are added to SAP's original loss. We expect the classification-based progress estimation to be more robust than regressing float values. Considering the range of $\text{dist}(\mathcal{S}, \mathcal{C}), \text{dist}(\mathcal{C}, \mathcal{A}), \text{dist}(\mathcal{A}, \mathcal{G})$, the normalizing constants D_1 and D_2 are set to 2 times the length of expert trajectory, and the length of expert trajectory, respectively.

In the finetuning stage, the agent performs sequential decision making in the scene. At each time step, the predicted action (a probability distribution on all the global candidates and "stop") is supervised through cross-entropy loss by a pseudo expert policy, which identifies the candidate node that minimizes the sum of the distances to the current node and the destination based on the complete graph of the scene. The agent then finds the shortest path from its current location to its chosen candidate on the graph it builds, and traverses along it to reach the next state. During finetuning, the agent chooses candidates by sampling from the fused action probability. While for inference, it selects the candidate with the maximum probability.

6.5. Model Statistics

In Table 5, we present the count of parameters for each module of our NavQ agent. Compared with DUET, the newly proposed FE and QM bring about 38% additional parameters, while they clearly boost the overall performance as shown in Table 1. Note that the Q-model is frozen when training the agent, reducing the impact on training cost. As for inference, we assess the efficiency by recording the average time for a forward pass of the full model. At each navigation step, DUET spends ~ 0.032s to make a decision, while NavQ spends ~ 0.052s under the same environment.

7. Details on the Datasets and Metrics

7.1. Datasets

Experiments are performed on two goal-oriented VLN datasets, REVERIE [96] and SOON [143]. REVERIE provides high-level descriptions of the target locations and objects as instructions. We adopt the same train/val/test split strategy as DUET [14]. The training set consists of 60 scenes and 10,466 instructions. The unseen validation set consists of 3,521 instructions in 10 scenes with no overlap to the training scenes. The test set consists of 16 novel scenes with 6,292 instructions. The average instruction length is around 21 words, and the expert trajectory typically requires 4–7 navigation steps. Pre-defined object bounding boxes are provided for each navigable location, and the agent needs to select one box as its predicted object. During training stage 2, We incorporate the additional synthetic instructions generated by a speaker model following DUET [14], which expand the training data from 10,466 to 30,102 instruction-path pairs.

SOON [143] is designed for a task named "From Anywhere to Object" (FAO). It requires the agent to find the target object no matter where its starting point is. The instructions are unrelated with the agent's initial location, but only describe the position and attributes of the target object, its relation to other objects, and its residing region. Each instruction contains an average of 47 words. The corresponding paths range from 2 to 21 steps. Object bounding boxes are not provided for SOON, and the agent must predict a direction representing the target object's center at the ending place of its trajectory. The training set of SOON comprises 3,085 instructions. Each instruction is paired with different starting points, resulting in 28,015 trajectories across 38 houses. The validation set and test set are composed of 339 instructions from 5 novel scenes, and 1,411 trajectories from 14 novel scenes. Each instruction is labeled with 10 different starting locations and 10 corresponding expert trajectories.

7.2. Evaluation Metrics

For navigation performance, we adopt the following standard metrics:

- **Success Rate (SR)**: The ratio of paths that successfully reach a correct location. For REVERIE, the correct locations are those where the target object is visible. For SOON, a ground-truth goal node is defined for each instruction by experts. The correct locations are the nodes within 3 meters of the goal node.
- **Oracle SR (OSR)**: The SR computed under an oracle

Table 6. An ablation study on the effect of Q-learning techniques. Results are obtained on REVERIE's val-unseen split.

Q-Model	OSR	SR	SPL	RGS	RGSPL
w.o.	54.42	48.14	33.38	30.19	21.05
vision-based	53.45	48.11	33.79	31.64	22.56
rand policy-based	58.68	51.29	36.23	34.34	24.59
ours	**60.47**	**53.22**	**38.89**	**36.84**	**27.12**

stop policy.

- **SR Penalized by Path Length (SPL)**: The SR adjusted to account for the path length. The original 0-1 success state is weighted by $\frac{\text{length of agent's traj}}{\text{length of expert's traj}}$.

We also utilize the following metrics that take object grounding into consideration:

- **Remote Grounding Success (RGS)**: The proportion of instructions executed successfully. For REVERIE, it requires the agent to output the correct object instance. For SOON, it requires that the output direction falls in the range of the correct object's bounding box.
- **RGS Penalized by Path Length (RGSPL)**: The RGS adjusted to consider the path length, similar to SPL.

8. Additional Experimental Results

8.1. Ablation Study on the Training of Q-Model

Here we give an analysis on the various techniques proposed in Section 3.3 for pre-training our Q-model. In Section 3.3.2, two designs are put forward for enhancing the generalizability of the Q-features. We have visualize the effect of the MAE pre-training by showing the loss curve in Figure 4, while the benefits of text-based prediction cannot be easily seen from the MSE loss, since the visual features and textual features have different scale. Thus, we compare using a visual prediction-based Q-model (*i.e.*, $\boldsymbol{R}$ set as the aggregated average view features) against our default setting. As is Table 6, employing textual features has a clear advantage over the vision-based Q-model.

Besides, we try out using a random policy instead of the π described in Section 3.3.1 and Section 6.2. For each state-action pair, we use simulations to approximate the expectation in Eq (6), where the agent uniformly chooses a local candidate at each rollout step. As in Table 6, integrating this random policy-based Q-model will lead to higher navigation performance than the baseline without Q-model, but the gain is less significant than our default setting. Therefore, the preference for optimal paths in π is indeed helpful for executing goal-oriented VLN tasks.

8.2. Results with Other Backbones

NavQ is a modular model enhancement that can be integrated with any baseline method focusing on leveraging historical information. In the main text, we mainly adopt DUET [14] as the baseline. In accordance with the reviewers' suggestions, here we explore an alternative backbone, BEVBert [2], which models the local environment with a top-down metric map, complementing the global topological representation. As shown in Table 7, incorporating the Q model and the future branch into BEVBert leads to notable improvements on most evaluation metrics, demonstrating the generalizability of the proposed method to some extent.

Table 7. The results on REVERIE with BEVBert as backbone.

		OSR	SR	SPL	RGS	RGSPL
Val	BEVBert [2]	56.40	51.78	36.37	34.71	24.44
Unseen	NavQ (Ours)	**60.07**	**54.08**	**38.49**	**35.36**	**25.45**
Test	BEVBert [2]	57.26	**52.81**	**36.41**	32.06	22.09
Unseen	NavQ (Ours)	**60.04**	52.42	36.40	**36.59**	**24.95**

Table 8. The results on R2R.

		TL	NE↓	SR↑	SPL↑
Val	DUET [14]	13.94	3.31	72	60
Unseen	NavQ	13.80	**3.06**	**73**	**63**
Test	DUET [14]	14.73	3.65	69	59
Unseen	NavQ	14.41	**3.30**	**72**	**63**

8.3. Results on Other Dataset

Based on the reviewers' suggestions, here we discuss the potential of NavQ on other VLN datasets. Apart from REVERIE [96] and SOON [143], there are some classical datasets, such as R2R [3] and RxR [56], in which the instructions are procedure-based rather than goal-based. As a result, the agent is required to follow the route described in the instructions, rather than merely reaching a specified destination. We note that our proposed method is tailored for *goal-oriented* VLN, and the formulation of Q-learning encourages the agent to reach the destination as quick as possible. Thus, NavQ is not quite suitable for procedure-based benchmarks, especially RxR, since it features non-shortest expert paths. We conduct preliminary experiments with NavQ on the R2R dataset, as it still satisfies the shortest-path assumption. As shown in in Table 8, NavQ achieves better performance than the base model, especially on the efficiency-related metric. However, the improvement is not as significant as on REVERIE, since the goal-centric future branch may not fully utilize the process-related information in the instructions.

8.4. More Visualization Results

In Figure 6, we provide two more qualitative comparisons between the NavQ agent and the baseline agent.

In addition, we discuss the distribution of navigation errors. Among the 3,521 validation instructions, our model produces 580 predicted trajectories that were identical to the expert trajectories, whereas DUET produces 468. For the remaining 2,941/3,053 trajectories, we analyze the position where the model makes the first error, *i.e.*, deviates from the expert trajectory. The results are presented in Figure 7. It

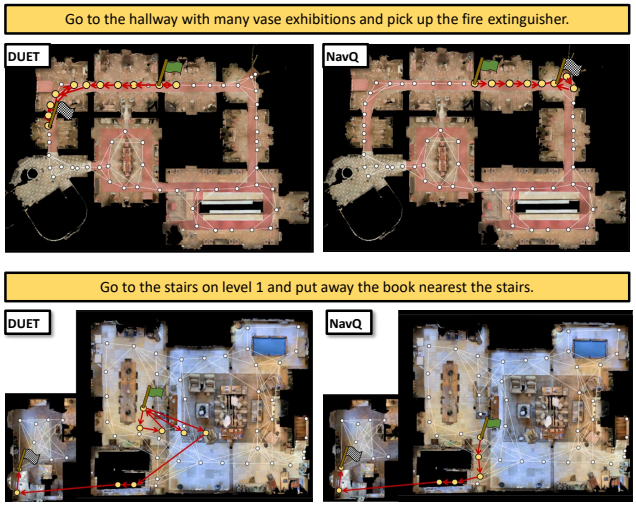

Figure 6. A qualitative comparison of our method and the baseline agent. In the upper example, NavQ reaches the correct destination while the baseline does not. In the lower example, NavQ arrives at the target object with less steps than DUET.

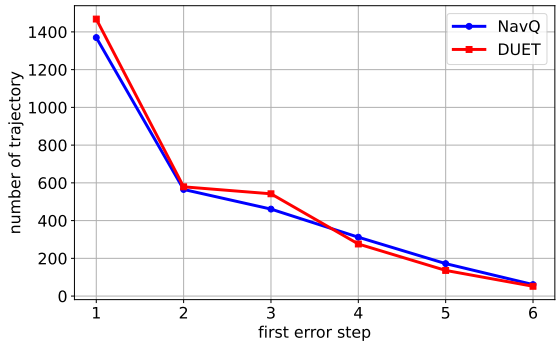

Figure 7. The distribution of navigation errors on REVERIE's val-unseen set.

can be noticed that our model makes fewer mistakes at the beginning and middle stage of the episode. This aligns well with the motivation of our foresighted agent, which is to make better decisions when the historical information (observations up to now) is not sufficient enough.