OVER-NAV: Elevating Iterative Vision-and-Language Navigation with Open-Vocabulary Detection and StructurEd Representation

Supplementary Material

7. System Prompt for GPT in Keyword Extraction

Here we provide the system prompt we use in keyword extraction in Table 6. After setting the system prompt, we send instructions to GPT and get responses containing the keywords separated by commas. Then we split the response and send the keywords for the following procedures.

System Prompt

You are a helpful assistant. You can help me by answering my questions. I will give you some instructions for vision-language navigation, you need to give me the key objects that are mentioned in this instruction. Key object is the noun or noun phrase that a navigation agent can use as milestone.
The query will be given by:
Instruction: ⟨QUERY⟩
You must respond to any queries or answer in the following way:
Query: ⟨QUERY⟩ Answer: ⟨ANSWER⟩ Therefore the answer is: ⟨TARGET_OBJETCTS⟩
The key objects in ⟨TARGET_OBJETCTS⟩ must appear in the instruction and are separated by commas.

Table 6. System Prompt for keyword extraction from instructions.

8. Iterative REVERIE

Here we further verify the effectiveness of our method on another navigation benchmark, REVERIE. REVERIE [36] is a benchmark for VLN with high-level instructions. The difference between REVERIE and R2R is that REVERIE replaces the instruction in R2R datasets with high-level instructions, which mainly describe the target location and objects. In contrast, R2R instructions provide detailed guidance to the agent along the ground truth navigation path.

To evaluate the agent under the iterative vision-and-language navigation setting of REVERIE, the benchmark needs to be transformed into the *iterative* version, which contains a tour file describing the episodes' order in which the instructions should be issued to VLN agents. Following [25], we generate the tours that minimize the distances between the end and starting points between the episodes in the tour. To this end, we employ Lin-Kernighan heuristic (LKH) , which is an efficient solver for the asymmetric travel salesman problem.

The comparison between our method, OVER-NAV and the baseline, HAMT is shown in Table 7. Our method can still achieve better performance in the challenging setting. Note that we report the performance of the checkpoint with the highest **t-nDTW** scores for both models, which is different from the original paper of HAMT[6].

9. Illustration of Two OVER-NAV Agents

9.1. OVER-NAV with HAMT

In Fig. 4, we present an overview of our method as applied to HAMT, the VLN agent for the discrete environment discussed in this paper. The illustration focuses on the data flow during step t of episode i. The HAMT part of Fig. 4 refers to IVLN[25].

The upper and lower sections of Fig. 4 depict OVER-NAV and HAMT, respectively. Within the HAMT framework, the language instruction for episode i undergoes processing in the instruction transformer, generating an embedding sequence of equal length, inclusive of the [CLS] and [SEP] tokens. Each embedding in the sequence corresponds to the instruction word in the same position. Concurrently, the agent captures observations during navigation, forwarding them to the vision transformer to extract image features. The ViT state feature, denoted as s_t^i, is produced by the vision transformer using both observations and angles. HAMT further maintains a history queue containing state-action pairs from previous steps within the current episode. The history transformer processes the history queue and generates the history embeddings. HAMT employs a cross-modal transformer encoder to fuse cross-modal inputs, where instruction embeddings serve as the text modal, while history and observation embeddings function as the visual modal. Notably, the instructions transformer, vision transformer, and history transformer undergo pre-training on proxy tasks before being frozen during the navigation task training in HAMT. Finally, the final model output is the action prediction a_t^i for the current step. HAMT then appends the state-action pair of the current step, s_t^i and a_t^i, to the history queue.

OVER-NAV prepares the keywords as described in Section 3. Subsequently, OVER-NAV leverages the same instruction transformer to derive embeddings for each keyword. Similar to the instructions, we add [CLS] and [SEP] tokens to each keyword, utilizing the embedding of the [CLS] token as the representation for the respective keyword. After the omnigraph fusion with the attached attributes, the keyword embeddings are arranged based on the distance metric d_i^t and are appended to the instruction

embeddings. The [SEP] token at the end of the instruction, serves as a separator between the two sections. Ultimately, the concatenated embedding functions as the text-modal representation and is transmitted to the cross-modal transformer encoder.

9.2. OVER-NAV with MAP-CMA

Fig. 5 illustrates the framework of our method when applied to MAP-CMA, the VLN agent in the continuous environment in this paper. The MAP-CMA part of Fig. 5 refers to IVLN [25].

The upper/bottom part of Fig. 5 shows MAP-CMA and OVER-NAV respectively. In MAP-CMA, the depth encoder encodes the depth images as depth embeddings and a bidirectional LSTM extracts the instruction embeddings from instructions. The segmentation map and occupancy map are concatenated and sent to the map encoder to produce the map embedding. The first GRU module serves as the state encoder, which encodes the depth embedding and map embedding at step t as the state embedding. The instruction attention is performed with instruction embeddings and state embedding to text embedding. Later the text embedding is sent to the visual attention module, which performs attention on the feature maps of depth images and map images. The second state encoder, *i.e.*, the GRU module, takes state embedding, text embedding, depth embedding, map embedding, and hidden state $h_{t-1}^{(a)}$ as inputs, and generates the predicted action a_t and new hidden state $h_t^{(a)}$ as outputs.

To incorporate OVER-NAV to MAP-CMA, we use the instruction bidirectional LSTM for keyword embedding extraction. Each keyword is represented by the [CLS] token embedding. After omnigraph fusion, the positional information, *e.g.*, heading and distance, is fused to the keyword embeddings. Then we perform the keywords attention with text embedding as the query. Finally, the omnigraph keyword context is sent to the second state encoder to aid the action prediction. Similar to Section 9.1, the omnigraph keyword context provides the distribution information of detected objects in previous episodes.

10. Code Implementation

We further provide the code implementation of our method in discrete environments, *i.e.*, on HAMT[6] in the supplementary material.

#	Model	Val-Seen					Val-Unseen				
		TL	OS ↑	SR ↑	SPL ↑	t-nDTW ↑	TL	OS ↑	SR ↑	SPL ↑	t-nDTW ↑
1	HAMT	9.8 ±0.2	22 ±1	23 ±1	20 ±1	38 ±1	9.6 ±0.1	20 ±2	22 ±1	19 ±0	28 ±1
6	Ours	9.8 ±0.3	**37** ±2	**40** ±1	**35** ±2	**44** ±1	8.9 ±0.2	**24** ±1	**25** ±2	**22** ±1	**30** ±1

Table 7. The comparison between HAMT and ours on REVERIE dataset.

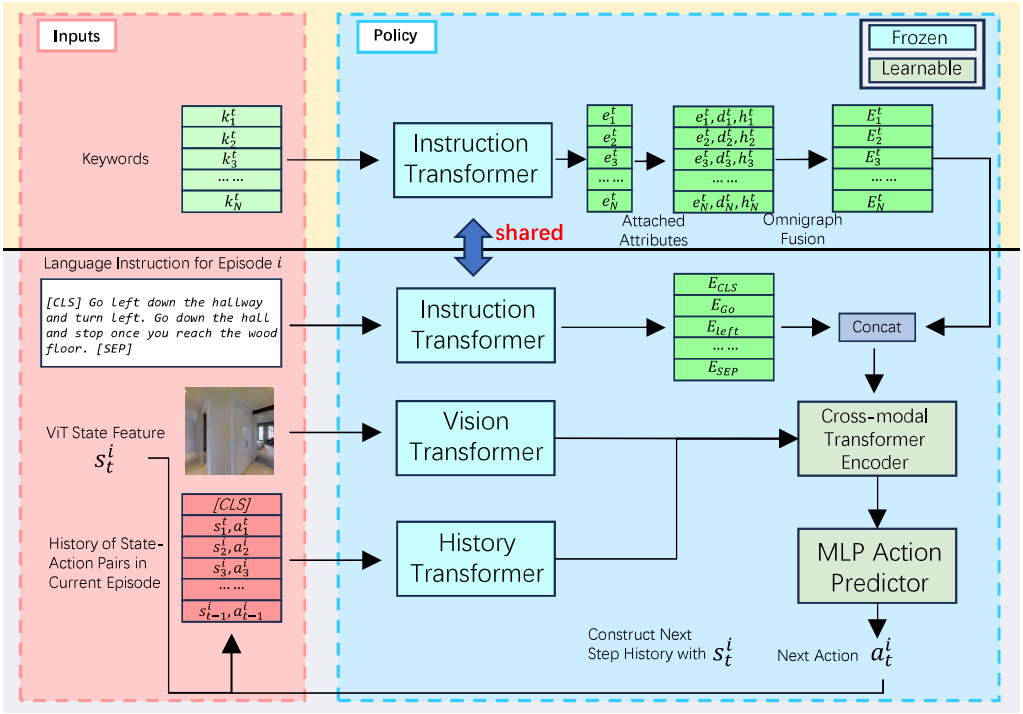

Figure 4. The overview of our method combined with HAMT.

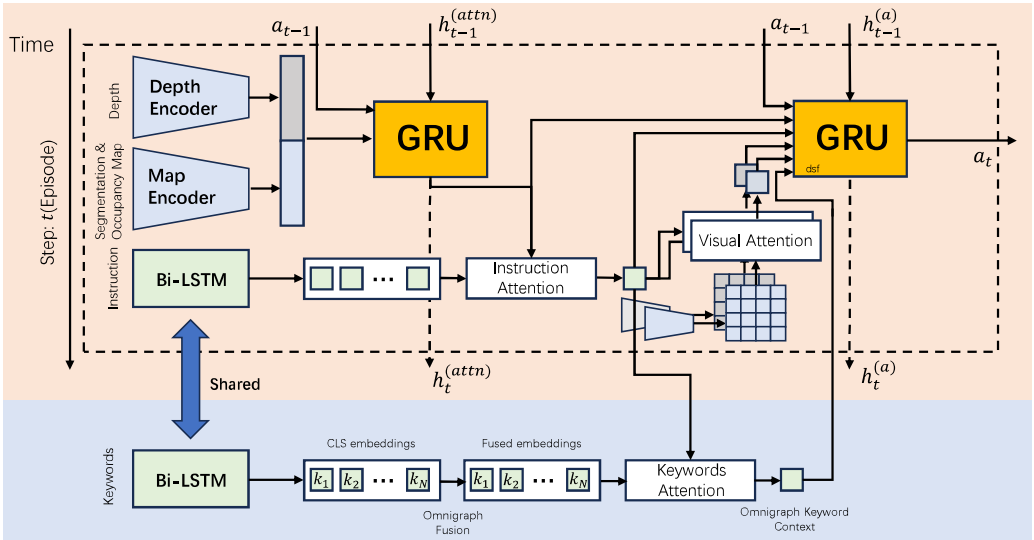

Figure 5. The overview of our method combined with MAP-CMA.