# Towards Long-Horizon Vision-Language Navigation: Platform, Benchmark and Method

Supplementary Material

## 1. Symbol Table

| Symbol | Explanation |
|---|---|
| $I_i$ | Observed image from **view** $i$ |
| $v_i$ | Visual features of **view** $i$ |
| $o_i$ | Fusion visual features of **view** $i$ |
| $S$ | Scene representation of **current observation** |
| $\mathcal{E}$ | Tokenizer |
| $h_i$ | History representation of **step** $i$ |
| $H_{n+1}$ | The memory set of the previous $n$ steps obtained at the $n + 1_{th}$ step |
| $\mathcal{G}$ | Large language model |
| $a_n$ | Model decision action at **step** $n$ |
| $e_n$ | Expert decision action at **step** $n$ |
| $n_v$ | The number of viewpoints in the observation. |
| $I_k$ | The indices of the $k$ selected elements |
| $M_{st}$ | Short term memory |
| $M_{lt}$ | Long term memory |
| $C$ | Confidence vector generated from $\mathcal{G}$ |
| $\mathcal{P}$ | Pooling function |

Table 1. Symbol Table

## 2. Benchmark

### 2.1. Trajectory Splitting Algorithm

We design a Trajectory Splitting Algorithm 1 for NavGen to backward-generate Step-by-Step Navigation Task from Navigation Trajectory.

### 2.2. Dataset Statistics

We conducted statistical analysis based on the complex task training set of the LH-VLN dataset, and the results are shown in Figure 1.

We also conducted statistical analysis on more detailed data regarding task goals at different stages in the dataset, as shown in Table. 2.

### 2.3. Extra Metric

We designed a metric Target Approach Rate (TAR) based on NE to reflect the model's performance in cases where the navigation success rate is relatively low. For the $i$-th subtask of the $j$-th task, we calculate the **Target Approach Rate (TAR)**:

$$tar_{j,i} = 1 - \frac{max(NE_{j,i} - D_s,\ 0)}{max(NE_{j,i}, GT_{j,i})} \tag{1}$$

where $NE_{j,i}$ is NE of the $i$-th subtask of the $j$-th task, $D_s$ is the distance to be considered success, $GT_{j,i}$ is ground truth of the $i$-th subtask of the $j$-th task.

**Algorithm 1:** Trajectory Splitting Algorithm

```
Input: Navigation trajectory $D_{traj}$, Image annotation model $Ram$
Output: Trajectory segments $Seg$
$actions = [turn\ left, turn\ right], s = []$;
for $action\ in\ actions$ do
    $i = 0$;
    while $i < D_{traj}.length - 3$ do
        $window = D_{traj}[i : i + 3]$;
        if $window.count(action) \geq 2$ then
            $indices = window.index(action)$;
            $s.append((index[0], index[-1], action))$;
            $i = index[-1] + 1$;
        else
            $i = i + 1$
$s.sort()$;
$merge\_s = [], c\_start, c\_end, c\_label = s[0]$;
for $s_i\ in\ s[1 :]$ do
    $start, end, label = s_i$;
    if $start \leq c\_end + 3\ and\ label == c\_label$ then
        $c\_end = max(c\_end, end)$;
    else
        $merge\_s.append((c\_start, c\_end, c\_label))$;
        $c\_start, c\_end, c\_label = s_i$;
$merge\_s.append((c\_start, c\_end, c\_label))$;
$Seg = [], last\_end = -1$;
for $strat, end, act\ in\ merge\_s$ do
    if $last\_end + 2 < start$ then
        $Seg.append((move\ forward, Ram(D_{traj}[last\_end + 1, start - 1]))$;
    $Seg.append((act, Ram(D_{traj}[start - 1, end + 1]))$;
    $last\_end = end$;
```

### 2.4. Cases Study

In the Figure 2, we present two cases for analysis.

For Case A, the agent quickly found the living room and identified the item on the table in Observation 3 as the target. However, in Observation 4, it determined that it was not a "bag", prompting a strategy change and further exploration of the scene (Observations 5–12). After completing the scene exploration, the agent resumed the search for the target.

For Case B, the agent was initially unable to determine the current scene and decided to rotate in place (Observa-

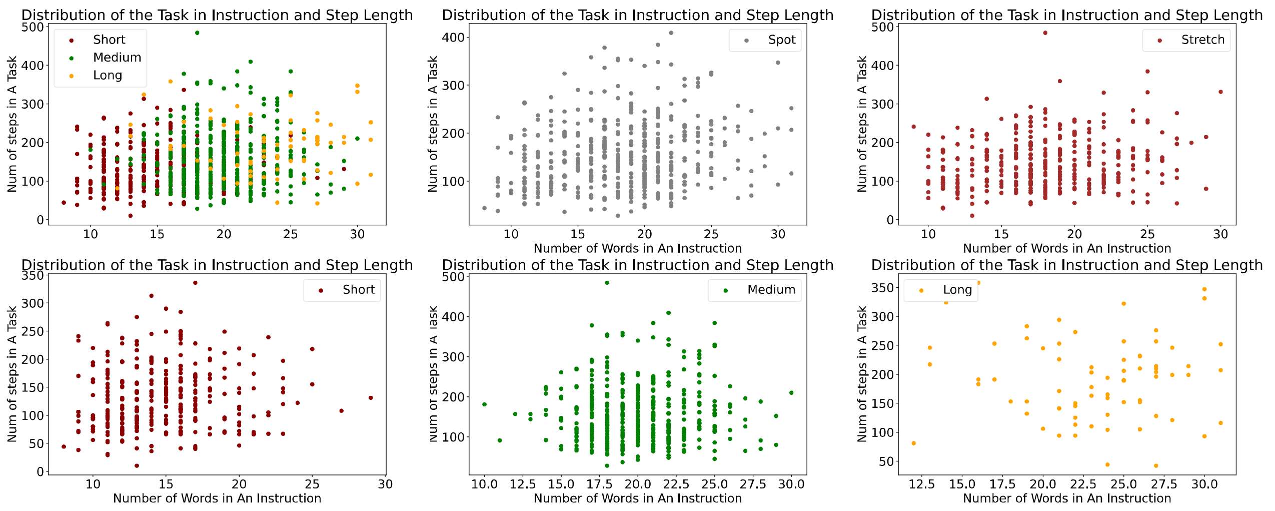


Figure 1. Statistic of the LH-VLN dataset distribution based on task length and robot configuration. We consider 2, 3, 4 subtasks as Short, Medium, and Long Task, respectively.

| Type | Task Steps | Nav Dis | Obj Num | Floor Span | Reentry Rate | Area Association |
|---|---|---|---|---|---|---|
| 2 subtasks | 68.39 | 11.60, 11.23, -, - | 288.43 | 1.05 | 27.44% | 1.02 |
| 3 subtasks | 53.30 | 10.32, 11.47, 4.74, - | 357.05 | 1.44 | 42.11% | 1.89 |
| 4 subtasks | 51.23 | 9.75, 10.45, 4.14, 9.39 | 273.37 | 1.88 | 48.84% | 2.12 |

Table 2. Detailed dataset statistics. *Task Steps* represents the average steps for each subtask. *Nav Dis* refers to the average geodesic distance between the agent and the target at the start of each stage, *Obj Num* is the average number of objects in the scene, *Floor Span* represents the average number of floors the task spans, *Reentry Rate* is the ratio of tasks where the agent needs to retrace its steps, indicating that the agent may have previously observed the target of the current subtask in prior tasks, and *Area Association* refers to the average number of identical subtask areas.

tions 1–4). It then tentatively searched for the living room. Upon realizing the current direction did not lead to the living room, it turned around (Observations 6–7) and moved toward the living room area (Observations 8–10). Since it did not find the "device" in the living room, the agent chose to explore another direction (Observations 11–12).

## 3. Extra Experiments

### 3.1. Prompt Used

We present our NavGen forward task generation prompt ($prompt_1$ 4), backward task generation prompt ($prompt_2$ 5), and MGDM model prompt ($prompt_3$ 6). $Prompt_3$ is designed with reference to [5].

### 3.2. Data Generation

We conduct experiments for data generation with NavGen in Habitat and Isaac Sim. Based on NavGen, we use HM3D scene assets and Habitat3's built-in greedy pathfinder algorithm to generate tasks and trajectories in Habitat3, and HSSD scene assets with a D* Lite-based [2] path planning algorithm to generate trajectories and tasks in Isaac Sim. Tasks and trajectories shown in the Figure 3.

### 3.3. More details on the experimental setup

Specifically, we designed two training approaches. One involves alternating between supervised learning and imitation learning at each step during the training of the main section. The other is the two-stage [5] training approach: pre-training with supervised learning for the first stage, followed by further training using reinforcement learning.

For supervised learning, we train on the trajectory dataset from the LH-VLN dataset. The agent obtains observations, location information, and action labels from the dataset without being directly deployed in the simulator. In imitation learning, the agent is deployed in the simulator, where it acts based on task instructions within the simulated environment. Expert actions are provided by Habitat 3's greedy pathfinder algorithm, and the agent learns to mimic these expert actions.

For additional experiments, we tested the state-of-the-art zero-shot model, InstructNav [3]. InstructNav decomposes instructions into dynamic chain-of-navigation using GPT-4. Based on DCoN, InstructNav optimizes navigation strategies from various perspectives using a simple numerical map and ultimately generates action decisions. Compared to other models, InstructNav utilizes additional information, including depth maps of the current viewpoint,

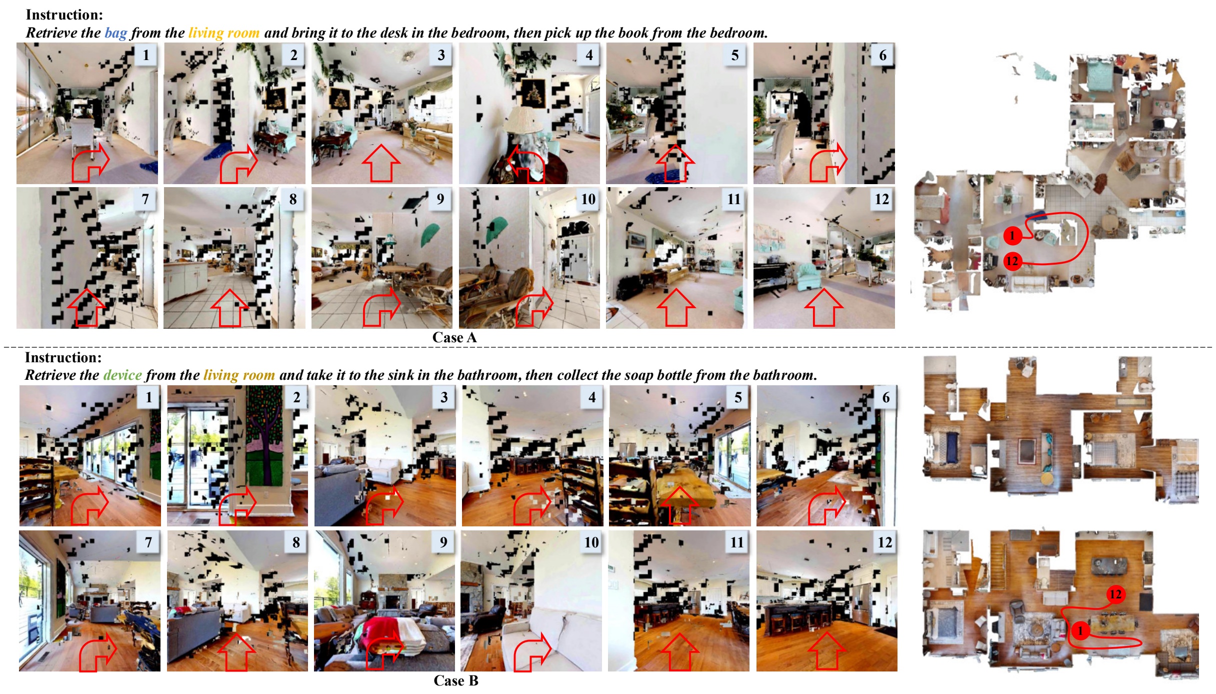
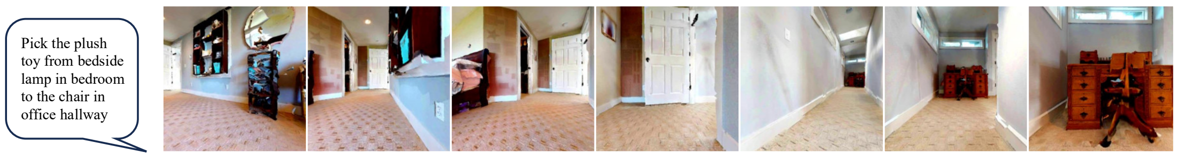
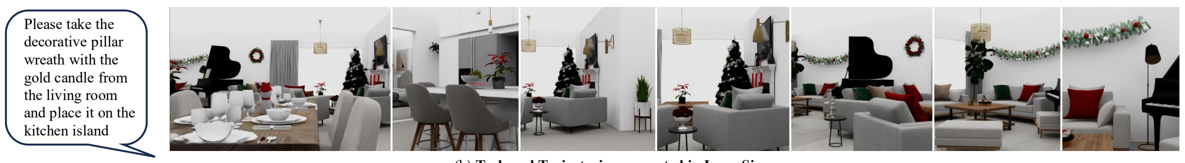

Instruction:
Retrieve the bag from the living room and bring it to the desk in the bedroom, then pick up the book from the bedroom.
1 2 3 4 5 6 7 8 9 10 11 12
Case A
Instruction:
Retrieve the device from the living room and take it to the sink in the bathroom, then collect the soap bottle from the bathroom.
1 2 3 4 5 6 7 8 9 10 11 12
Case B

Figure 2. Part of trajectories of two Task.


Pick the plush toy from bedside lamp in bedroom to the chair in office hallway

(a) **Task and Trajectories generated in Habitat**


Please take the decorative pillar wreath with the gold candle from the living room and place it on the kitchen island

(b) **Task and Trajectories generated in Isaac Sim**

Figure 3. Tasks and trajectories generated with NavGen in Habitat and Isaac Sim.

top-down maps of the scene, and partial use of Habitat 3's greedy pathfinding interface.

Furthermore, we replaced the large language model in MGDM to test its compatibility with different large models and the impact of these models on test results, following the two-stage training approach. The model used for replacement testing is Llama 3.1 8B Instruct [4]. Due to GPU memory constraints, we froze the first five layers of the model's language layers.

## 3.4. More experimental results

First, we supplement the test results under different robot configurations (Spot, Stretch) in Table 3.

Based on the Random results, the difficulty of tasks executed by the Spot and Stretch robots differs. The NE (Navigation Error) for Spot is greater than that for Stretch, indicating that the agent, on average, is farther from the objects in tasks involving the Spot robot. Under this premise, the data in the table 3 shows that, apart from the zero-shot model InstructNav and GLM-4v prompt, which perform relatively evenly across both types of tasks, other models generally perform better on Spot robot tasks compared to Stretch robot tasks. Given that the LH-VLN training set has a relatively balanced distribution of both types of tasks, and the NaviLLM pretrain model (which was not trained on the LH-VLN dataset) exhibits the same trend, we believe the performance difference may stem from the lower viewpoint of the Spot robot. This configuration excludes the influence of smaller objects, making it easier for the agent to approach the target area.

Additionally, in tasks with the Spot robot configuration, MGDM's performance is slightly lower than

| Method | Spot | | | | | Stretch | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SR↑ | NE↓ | ISR↑ | CSR↑ | CGT↑ | SR↑ | NE↓ | ISR↑ | CSR↑ | CGT↑ |
| Random | 0. | 14.97 | 0. | 0. | 0. | 0. | 10.48 | 0. | 0. | 0. |
| InstructNav [3] | 0. | 11.48 | 0. | 0. | 0. | 0. | 7.70 | 0. | 0. | 0. |
| GLM-4v prompt [1] | 0. | 15.97 | 0. | 0. | 0. | 0. | 10.55 | 0. | 0. | 0. |
| NaviLLM (Pretrain) [5] | 0. | 10.27 | 0. | 0. | 0. | 0. | 11.43 | 0. | 0. | 0. |
| NaviLLM [5] | 0. | 10.97 | 2.19 | 1.31 | 2.72 | 0. | - | 0 | 0 | 0 |
| GPT-4 + NaviLLM | 0. | 10.15 | **3.85** | **2.00** | **3.96** | 0. | 11.59 | 2.33 | 1.59 | 2.47 |
| **MGDM (Ours)** | 0. | **3.44** | 3.73 | 1.92 | 3.81 | 0. | **1.37** | **2.82** | **2.06** | **3.25** |

Table 3. Performance of SOTA models in LH-VLN Task under different Robot configurations.

| Method | Type | 2-3 Subtasks | | | | | 3-4 Subtasks | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | SR↑ | NE↓ | ISR↑ | CSR↑ | CGT↑ | SR↑ | NE↓ | ISR↑ | CSR↑ | CGT↑ |
| Random | - | 0. | 14.09 | 0. | 0. | 0. | 0. | 10.91 | 0. | 0. | 0. |
| InstructNav [3] | Zero-shot | 0. | 10.80 | 0. | 0. | 0. | 0. | 8.38 | 0. | 0. | 0. |
| GLM-4v prompt [1] | Zero-shot | 0. | 15.63 | 0. | 0. | 0. | 0. | 10.97 | 0. | 0. | 0. |
| NaviLLM [5] | Pretrain | 0. | 12.11 | 0. | 0. | 0. | 0. | 10.04 | 0. | 0. | 0. |
| NaviLLM [5] | Finetuned | 0. | 12.24 | 0. | 0. | 0. | 0. | 9.79 | 3.54 | 2.53 | 5.24 |
| GPT-4 + NaviLLM | Pretrain | 0. | 12.23 | 0. | 0. | 0. | 0. | 10.00 | 4.37 | 2.91 | 5.23 |
| MGDM (Llama3/Two Stage) | Finetuned | 0. | 13.20 | 0. | 0. | 0. | 0. | 10.02 | 4.60 | 3.07 | 5.38 |
| MGDM (Vicuna/Alternate) | Finetuned | 0. | **3.54** | 0. | 0. | 0. | 0. | **1.23** | 4.69 | 3.30 | 5.83 |
| MGDM (Vicuna/Two Stage) | Finetuned | 0. | 10.44 | 0. | 0. | 0. | 0. | 8.78 | **5.13** | **3.42** | **6.00** |

Table 4. Performance comparison in LH-VLN Task with different task length. We add InstructNav, MGDM (Llama3/Two Stage), MGDM (Vicuna/Two Stage) for r comparison. (Vicuna/Two Stage) stands for LLM and training set used.

NaviLLM+GPT-4. Notably, for tests involving NaviLLM-related models, their performance shows a significant advantage under the Spot robot configuration. This suggests that the NaviLLM model is relatively better suited for Spot-related tasks. This advantage may stem from NaviLLM's pretraining data being more closely aligned with the Spot robot configuration, giving it a certain edge in this aspect. In contrast, MGDM, due to the design of its memory module, is more likely to "remember" small objects in the scene. This may reduce the performance gap between the Spot and Stretch robot configurations, leading to more balanced results across both setups.

Furthermore, we include InstructNav from the additional experiments, as well as MGDM models employing different large language models and training strategies, for comparison. As shown in Table 4, due to its Dynamic Chain-of-Navigation module's ability to comprehend and decompose complex tasks, InstructNav performs exceptionally well among zero-shot models. However, some of InstructNav's configurations, such as trajectory value maps that avoid historical paths, are not well-suited for multi-stage complex tasks.

The impact of replacing Llama 3 is limited, which may be attributed to the performance differences between Vicuna 1.5 and Llama 3, as well as the effects of varying training configurations on the model. MGDM trained with the two-stage setup achieved better performance on ISR, CSR, and CGT metrics compared to alternating training but performed relatively worse on the NE metric. This suggests that different training setups were not effective in addressing the model's difficulty in determining whether the target has been successfully reached.

For the issue where tasks with fewer subtasks yield worse results, we consider the following reasons:

The first reason that the model performs worse in shorter tasks is that the model needs sufficient steps to warm up, i.e., setting up correct initial direction and locate targets. Though shorter tasks may seem easier, they leave fewer steps after initialization, making it more challenging for the agent to complete the task, thus leading to worse performance.

In our dataset analysis, as the number of subtasks in complex tasks increases, the average ground truth steps per subtask decrease (68.39, 53.30, 51.23), while the average maximum number of identical regions in the task increases (1.02, 1.89, 2.12), the probability of the agent observing critical information about subsequent subtasks during current subtasks increases(27.44%, 42.11%, 48.84%). This suggests that with more subtasks, the probability of different subtask targets appearing in the same region increases, the average number of steps required for each subtask decreases, and the average difficulty of each subtask decreases.

This is also reflected in the distances between subtask goals. In Table. 2, the navigation distance for the third subtask (i.e., the distance between the second and third goals) is always significantly smaller than the navigation distances at other stages. (4.74, 4.14 to 10.32, 11.47 and 9.74, 10.45) This indicates that in complex tasks with three or more subtasks, once the second goal is found, it becomes much easier to locate the third goal.

**System**:
You are proficient in planning and design. You need to design a practical task consisting of multiple navigation-interaction subtasks based on the scene(which contain a large number of objects) and robot characteristics.
**Rules**:
There are two part of the input: scene and robot. The scene includes objects in different regions of the scene. These objects serve as the foundation for generating your task, your task must be based on the scene. And the input robot includes the charactor of the robot, which is what you need to consider when you generate task.
The task you generate should consist of the following three subtasks:
""""
- Move_to("object_region id"): Walk to an object in a region. PAY ATTENTION that "object_region id" SHOULD be composed of the object and the ID of the corresponding region of the object in the input scene.
- Grab("object"): When the robot move to an object, it may need to grab it. To perform it, the robot need to move to the object first and make sure the robotic arm is empty. "object" should correspond to the object to be grab. PAY ATTENTION that "object" SHOULD be in the input scene.
- Release("object"): When the robot move to a place and had an object in hand, it can release the object in hands to the place. To perform it, the robot need to move to the place first and make sure the robotic arm is not empty. "object" should correspond to the object to be released. PAY ATTENTION that "object" SHOULD be in the input scene.
""""
There are something you need to pay attention to:
""""
- Avoid words like "grab" and "release" in instructions.
- Pay attention that the objects in your task should be limited in 1 to 2 regions. And you should mention these regions in instruction. All of these regions SHOULD be in input scene.
- The task you generate should be similar to instructions like "Take an object in one region to a certain place in one region, then retrieve an object from that place."
- The object you take/grab should be portable.
- You need to generate a task consist of 4 to 6 subtasks.
- The region id in subtask "Move_to" SHOULD be corresponded to the region in instruction.
""""
Your output should be a python dictionary which has two keys:
- dictionary["Task instruction"]: A conversational task instruction to describe the task.
- dictionary["Subtask list"]: A list of subtasks that make up the task.
Make sure the task instruction conversational enough, and the task should reasonable.

**Example**:
Here is an example of the INPUT and OUTPUT:
INPUT:
"'
Scene: "Region 0: Bedroom": ["bookshelf", "toy", "lamp", "whiteboard", "bed", "nightstand", "hanging clothes", "picture", "rug", "bag", "chest of drawers", "basket of something", "box", "clothes rack", "shoe"], "Region 3: Bathroom": ["door", "toilet", "bin", "toilet brush", "picture", "soap dispenser", "toilet paper", "sink", "sink cabinet"], "Region 4: Office": ["picture", "stationery", "statue", "ornament", "box", "folder", "printer", "stool", "bin", "plant", "computer", "mouse", "keyboard", "remote control", "chair", "water dispenser", "cushion", "desk", "light fixture"]
Robot: "Spot in simulator is an agile, quadrupedal robot. Action: Spot support three kinds of action: move_forward, turn_left and turn_right. Sensors: Spot is equipped with three RGB cameras at a height of 0.5 meters in front, left and right to obtain embodied images in these three directions. Mobility: Spot's four-legged design allows it to navigate challenging terrains, including stairs, rocky surfaces, and cluttered environments. Use: As spot is shaped like a dog, it can do some work for human in domestic scenes. It has a simple robotic arm, but is positioned lower (0.5 meters) and can perform some simple grabbing."
"'
OUTPUT: "' "Task instruction": "take the bag in bedroom to the desk in office", "Subtask list": ["Move_to('bag_0')", "Grab('bag')", "Move_to('desk_4')", "Release('bag')"]

Figure 4. $prompt_1$ for forward task generation

**System**:
You are good at guiding the way. Now you need to generate step-by-step navigation instructions based on some information.

**Rules**:
The INPUT is a dictionary, the format is as follows:
‘‘‘
{”target”: The target of navigation,
”step_1”: A dictionary containing the action of this step and the scene tags related to the environment of this step,
...
”step_n”: A dictionary containing the action of this step and the scene tags related to the environment of this step.}
‘‘‘
You need to combine the action of each step and the corresponding environment description scene tags based on the INPUT, and finally get a complete, easy-to-understand, and accurate navigation instruction. Please note the following requirements:
- You only need to select one most appropriate scene tag for each step to form a smooth and logical navigation instruction.
- The step corresponding to the “move_forward” action should preferably select a specific scene from the scene tags.
- The step corresponding to the “turn_left” and “turn_right” actions should preferably select a specific object from the scene tags.
- The order of each action in the final instruction should be the same as the order of steps in the input, the total number of steps in the instruction should be the same as the number of steps in the INPUT, and the scene-related information involved should be in the scene tags of the corresponding action.
- The last step of the instruction should contain the navigation target, that is, the target in the INPUT.
- Your output should only be the instruction. The instruction should not contain words such as “step”.

**Example**:
Here is an example:
‘‘‘
INPUT: {’target’: ’guitar case’, ’step_0’: ’action’: ’move_forward’, ’tags’: [’store’, ’retail’, ’shopper’, ’mall’, ’fill’], ’step_1’: ’action’: ’make a left turn’, ’tags’: [’equipment’, ’room’, ’store’, ’fill’, ’retail’], ’step_2’: ’action’: ’move_forward’, ’tags’: [’beam’, ’retail’, ’store’, ’warehouse’, ’room’], ’step_3’: ’action’: ’turn_right’, ’tags’: [’retail’, ’room’, ’store’, ’warehouse’, ’fill’], ’step_4’: ’action’: ’move_forward’, ’tags’: [’room’, ’store’, ’fill’, ’shelf’, ’retail’], ’step_5’: ’action’: ’turn_left’, ’tags’: [’electronic’, ’equipment’, ’fill’, ’room’, ’store’], ’step_6’: ’action’: ’move_forward’, ’tags’: [’retail’, ’shelf’, ’store’, ’fill’, ’room’]}
OUTPUT:
Move forward through the store and make a left turn at the equipment, go ahead through the store, turn right at the retail shelf, move forward and turn left by the electronic equipment, finally go straight to the guitar case.
‘‘‘
Please get the output according to the above prompts and the following INPUT. Think step by step.

Figure 5. $prompt_2$ for backrward task generation

**”Task”**: ”Instruction: Go to the locations to complete the given task. Task: ”,
**”History”**: ”Following is the History, which contains the visual information of your previous actions.”,
**”Observation”**: ”Following is the Candidate, which contains several directions you can go to at the current position, candidate (0) is stop.”,
**”Output Hint”**: ”Compare the History and Instruction to infer your current progress, and then select the correct direction from the candidates to go to the target location and finish the task.”

Figure 6. $prompt_3$ for MGDM.

## References

[1] Team GLM, Aohan Zeng, Bin Xu, and Zihan Wang. Chatglm: A family of large language models from glm-130b to glm-4 all tools, 2024. 4

[2] Sven Koenig and Maxim Likhachev. D*lite. In *AAAI/IAAI*, 2002. 2

[3] Yuxing Long, Wenzhe Cai, Hongcheng Wang, Guanqi Zhan, and Hao Dong. Instructnav: Zero-shot system for generic instruction navigation in unexplored environment. *CoRR*, abs/2406.04882, 2024. 2, 4

[4] Meta. Llama 3, 2024. 3

[5] Duo Zheng, Shijia Huang, Lin Zhao, Yiwu Zhong, and Liwei Wang. Towards learning a generalist model for embodied navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13624–13634, 2024. 2, 4