

SayNav: Grounding Large Language Models for Dynamic Planning to Navigation in New Environments

Primary Keywords: *Learning; Robotics; Knowledge Representation/Engineering*

A Multi-Object Navigation Dataset

We introduce a benchmark dataset for multi-object navigation for our evaluation and future use by researchers. We use houses from ProcThor-10K (Deitke et al. 2022) dataset to build a photo-realistic MultiON benchmark dataset of 132 episodes. We have provided the dataset file in the supplementary material where each episode is described using the following properties:

1. *data_type* : This corresponds to either ‘val’ or ‘test’ based on which set of data was used from ProcThor-10K to construct the episode.
2. *house_idx* : This corresponds to index of the specific house in the ProcThor-10K dataset.
3. *num_rooms* : Number of rooms in the house
4. *num_targets* : Number of targets in the house (Currently we have limited the dataset to 3 targets)
5. *targets* : List of unique targets in the house along with their ground truth locations
6. *start_position* : Start position randomly sampled from all reachable positions in the house
7. *start_heading* : Start heading randomly sampled from 0, 90, 180 and 270 degrees
8. *shortest_path_targets_order* : The order of targets that results in shortest path using A* planner. This is evaluated by running A* planner on all possible target orders.
9. *shortest_path_length* : Length of the shortest path computed via A* planner along the *shortest_path_targets_order*

B Additional Implementation Details

We provide additional implementation details for both the high-level and the low-level planner.

B.1 High-Level Planner

Figure 1 shows the structure of prompts used for identifying the room type and determining the feasibility of locating target objects in a room.

B.2 Low-Level Planner

For the low-level POINTNAV planner, we first implement a shortest path oracle using A* algorithm on the grid of reachable positions in the house, and then train a POINTNAV agent using the DAGGER (Ross et al. 2011) algorithm with the A* oracle as the expert.

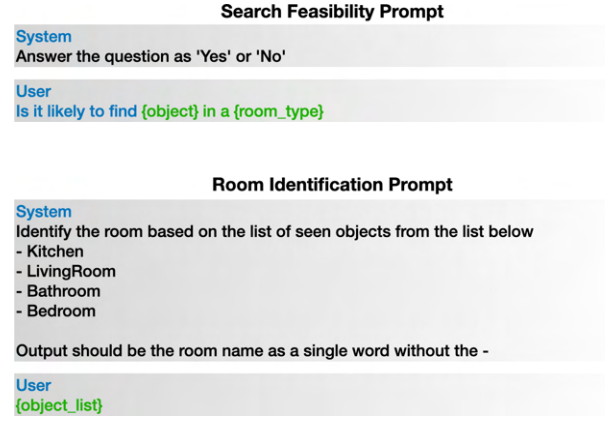


Figure 1: Prompt used to compute the feasibility of finding an object in a particular room as well for identifying which room the agent is in.

We perform DAGGER dataset aggregation over two rounds. In the first round, 2,000 episodes were collected using a random agent. In the second round, 800 additional episodes were collected using an agent trained using episodes from the first round. The aggregated dataset contains a total of 2,800 episodes or 7×10^5 simulator steps for IL. For each episode, we choose a scene from the ProcThor-10k train split, randomly sample a start location and sample a random object from the scene as the goal location. The robot then performs the task by taking expert action with $p = 0.2$ and agent action from $p = 0.8$. The robot’s observations and expert actions are stored for behavior cloning.

For behavior cloning, the objective function is to minimize cross entropy loss of predicted action against expert action prediction at every step. For agent architecture, following (Wijmans et al. 2019), we use a standard architecture shown in Figure 2. As discussed in the Approach section, our agent receives an RGBD image and the agent’s pose with respect to the goal location as the input. A GroupNorm (Wu et al. 2018) ResNet18 (He et al. 2016) encodes the input RGBD image, and a 2-layer 512 hidden size gated recurrent unit (GRU) (Cho et al. 2014) combines history and sensor inputs to predict the next action. We use the Adamax optimizer with $\text{lr } 10^{-4}$ and weight decay 10^{-4} . We optimize

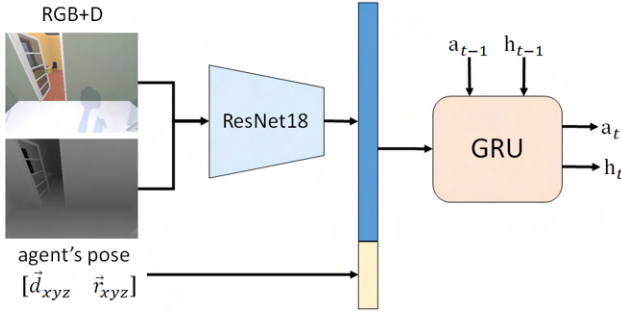


Figure 2: Low-level POINTNAV architecture.

the objective function with batch size of 16 episodes for 50 epochs.

We evaluate the POINTNAV performance of the agent on the publicly available AI2Thor OBJECTNAV dataset¹ ProcThor-10k-val split, given the ground truth location of the object as the goal for POINTNAV evaluation. The evaluation split contains 1550 episodes. When multiple instances of the target object are available, we arbitrarily select the first instance as the POINTNAV goal. Our low-level planner achieves 84.5% POINTNAV success rate (success radius 1.5m, max 300 steps) and 0.782 SPL. On the subset of episodes where the starting position and the goal position are in the same room, performance increases to 98.5% success rate and 0.930 SPL.

C Memory via LLM

As mentioned in the primary manuscript, SayNav uses the 3D scene graph to support memory for future planning. For instance, it automatically annotates the room nodes that have already been investigated and won't plan for the room if the agent happens to visit the same room again. This type of implementation to support memory works perfectly for our chosen task. However, we also wanted to explore the possibility of making the LLM track its own plans. Hence, we also implemented SayNav's memory via LLM by using Conversational Chains module of the LangChain framework². Here we use two separate instances of identical LLM models, one to generate the high level plans and another to track the generated plans. The LLM tracking the generated plan, uses the **Room Tracking Prompt** in Figure 3 and is also equipped with a conversational memory. The LLM responsible for generating the plans receives detailed description of the surrounding environment while the other LLM instance only receives the minimal information necessary to do the tracking. A key advantage of this framework is that it avoids hitting the maximum token limit on the LLM by not relying on the entire conversational history (as is usually done). We believe that LLM-based tracking might be able to generalize better to other tasks since it eliminates the need of a module for tracking plan history, which would require different implementations for different task (we will test this

¹<https://github.com/allenai/object-nav-eval>

²https://python.langchain.com/docs/modules/chains/how_to/memory

in our future work).

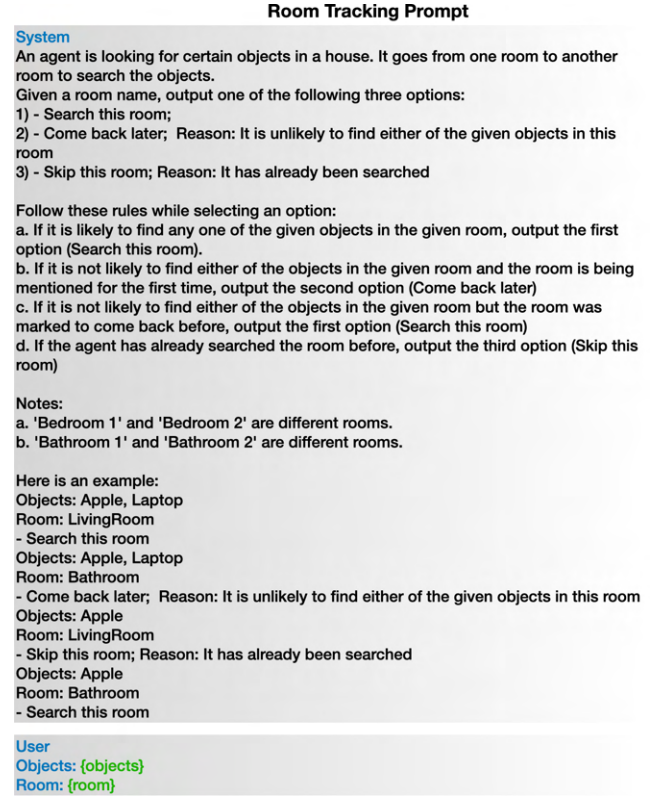


Figure 3: Prompt used to compute the feasibility of finding object(s) in a particular room as well for tracking the rooms explored by the agent.

Table 1 shows the performance of SayNav using LLM Memory via *gpt-3.5-turbo* and *gpt-4*. Note that we use the same model for both the instances of LLM in our experiments. We do observe substantial drop in SR and SPL metrics by using LLM Memory in the case of *gpt-3.5-turbo* as compared to the results reported in the main article. However, with *gpt-4*, the LLM Memory is able to achieve similar results as compared to the results reported in the main article. This shows that it is feasible to hand-over the task of tracking to the better LLM models.

D Additional Qualitative Results

We have provided a video demo for our agent exploring one of the multi-room houses. For the same episode we have shown the log of outputs generated by the agent in Figure 4, 5, and 6.

References

Cho, K.; et al. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *abs/1406.1078*. ArXiv.

Deitke, M.; et al. 2022. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. *NeurIPS*, 35: 5982–5994.

0. Entered a new room

1. Looking around

2. It is unlikely to find AlarmClock, Laptop, CellPhone in Kitchen, So will come back later

3. Could not complete goal, so going into exploration mode: Go to a Door

4. Walking to the door

5. Walk into the neighboring room

6. Entered a new room

7. Looking around

8. **Found:** Laptop

9. Executing
action=navigate
arg=(4.59, 6.33)
Go to TVStand
Currently in room: LivingRoom

10. Executing
action=look
arg=AlarmClock, CellPhone
Look for AlarmClock and CellPhone near the TVStand
Currently in room: LivingRoom

11. **Found:** CellPhone

12. Executing
action=navigate
arg=(5.36, 3.72)
Go to DiningTable
Currently in room: LivingRoom

13. Executing
action=look
arg=AlarmClock, CellPhone
Look for AlarmClock and CellPhone near the DiningTable
Currently in room: LivingRoom

14. Executing
action=navigate
arg=(3.61, 6.36)
Go to Dresser
Currently in room: LivingRoom

15. Executing
action=look
arg=AlarmClock, CellPhone
Look for AlarmClock and CellPhone near the Dresser
Currently in room: LivingRoom

16. Could not complete goal, so going into exploration mode: Go to a Door

17. Walking to the door

18. Walk into the neighboring room

Figure 4: Example of SayNav's sequential plan

19. Entered a new room

20. Looking around

21. Executing
action=navigate
arg=(0.67, 1.21)
Go to Bed
Currently in room: Bedroom 1

22. Executing
action=look
arg=AlarmClock
Look for AlarmClock near the Bed
Currently in room: Bedroom 1

23. Executing
action=navigate
arg=(0.22, 4.34)
Go to Dresser
Currently in room: Bedroom 1

24. Executing
action=look
arg=AlarmClock
Look for AlarmClock near the Dresser
Currently in room: Bedroom 1

25. Executing
action=navigate
arg=(2.37, 4.93)
Go to DiningTable
Currently in room: Bedroom 1

26. Executing
action=look
arg=AlarmClock
Look for AlarmClock near the DiningTable
Currently in room: Bedroom 1

27. Executing
action=navigate
arg=(2.37, 5.97)
Go to Chair1
Currently in room: Bedroom 1

28. Executing
action=look
arg=AlarmClock
Look for AlarmClock near Chair1
Currently in room: Bedroom 1

29. Executing
action=navigate
arg=(2.37, 3.88)
Go to Chair2
Currently in room: Bedroom 1

30. Executing
action=look
arg=AlarmClock
Look for AlarmClock near Chair2
Currently in room: Bedroom 1

Figure 5: Example of SayNav's sequential plan (cont'd)

<p>31. Executing action=navigate arg=(2.97, 4.41) Go to Chair3 Currently in room: Bedroom 1</p> <p>32. Executing action=look arg=AlarmClock Look for AlarmClock near Chair3 Currently in room: Bedroom 1</p> <p>33. Executing action=navigate arg=(2.97, 5.4) Go to Chair4 Currently in room: Bedroom 1</p> <p>34. Executing action=look arg=AlarmClock Look for AlarmClock near Chair4 Currently in room: Bedroom 1</p> <p>35. Executing action=navigate arg=(1.62, 7.87) Go to Safe Currently in room: Bedroom 1</p> <p>36. Executing action=look arg=AlarmClock Look for AlarmClock near the Safe Currently in room: Bedroom 1</p> <p>37. Executing action=navigate arg=(0.61, 7.56) Go to DogBed Currently in room: Bedroom 1</p> <p>38. Executing action=look arg=AlarmClock Look for AlarmClock near the DogBed Currently in room: Bedroom 1</p> <p>39. Could not complete goal, so going into exploration mode: Go to a Door</p>	<p>40. Walking to the door</p> <p>41. Walk into the neighboring room</p>	<p>42. Entered a new room</p> <p>43. Looking around</p> <p>44. Found: AlarmClock</p>
--	--	---

Figure 6: Example of SayNav’s sequential plan (cont’d)

	Scene Graph	LL Planner	SR (%)	SPL	Kendall Tau
SayNav (gpt-3.5)	GT	OrNav	77.86	0.37	0.72
	GT	PNav	61.83	0.24	0.76
	VO	OrNav	58.73	0.40	0.72
	VO	PNav	46.77	0.29	0.82
SayNav (gpt-4)	GT	OrNav	93.93	0.43	0.69
	GT	PNav	86.36	0.35	0.77
	VO	OrNav	72.09	0.44	0.73
	VO	PNav	61.60	0.35	0.77

Table 1: Results of SayNav on multi-object navigation task using LLM Memory

He, K.; et al. 2016. Deep residual learning for image recognition. In *CVPR*. IEEE.

Ross, S.; et al. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS. JMLR Workshop and Conference Proceedings*. 130

Wijmans, E.; et al. 2019. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *abs/1911.00357*. ArXiv.

Wu, Y.; et al. 2018. Group normalization. In *ECCV*, 3–19. 135