8 Supplementary Material

This section will elaborate on some details. The table of contents is as follows.

- Demand-Driven Navigation Dataset 8.1
  - Generation Process 8.1.1
  - Statistical Features 8.1.2
- Demand-Driven Navigation Method 8.2
  - Textual Attribute Feature Learning 8.2.1
    - Language-grounding Mappings 8.2.1
    - t-SNE of Demand-conditioned Attribute Feature 8.2.1
  - Policy Learning 8.2.2
    - Pipeline in Transformer 8.2.2
    - Trajectory Collection 8.2.2
    - Demand-based Visual Grounding Model 8.2.2
  - Model Pre-training and Fine-tuning(8.2.3
    - Image Encoder: Vision Transformer 8.2.3
    - DETR 8.2.3
- Experiments 8.3
  - Metrics 8.3.1
  - Baselines 8.3.2
    - VTN-object, VTN-demand, VTN-GPT 8.3.2
    - ZSON-object, ZSON-demand, ZSON-GPT 8.3.2
    - GPT-3+Prompt, MiniGPT-3 8.3.2
- Ethics 8.4

8.1 Demand-Driven Navigation Dataset

8.1.1 Generation Process

In order to generate the world-grounding mappings, which is referred to as the DDN dataset, for the room dataset used in our experiments, we utilise Prompts Engineering and query GPT-3 with the prompts provided below.

> I have some objects. Here are the list of them.
> $object_1, object_2, object_3$.....
> Question: What human demand can these objects satisfy?
> Tips: Please follow the format of these examples shown below. A demand can be satisfied by different objects. These objects are interchangeable with each other. Objects must strictly meet demand. If there are multiple objects that satisfy the demand, separate them with commas.
> Examples:
> Apple, Egg: I am so hungry. Please give me some food.
> Bed, Sofa, Stool, Armchair: I am tired and need a place to rest.
> Microwave: I wish to heat up the food.
> Safe: I hope to find a place to store my valuables.

After using Prompts Engineering to query GPT-3, the objects and corresponding demands are outputted in the format of our given examples, and subsequently saved as a JSON file in dictionary format. Since the world-grounding mappings generated by GPT-3 are occasionally inaccurate, all authors perform manual filtration and supplementation. Inclusion of objects in the final world-grounding mappings is determined by agreement amongst all authors regarding its ability to satisfy a demand. Here are some of the generated world-grounding mappings:

- "I would like to know what time it is.": "AlarmClock", "CellPhone", "Laptop", "Desktop", "Watch"
- "I need a container to store liquids.": "Bottle", "Mug", "Cup", "WineBottle", "Kettle"
- "I am hungry, please give me something to eat with.": "Apple", "Egg", "Potato", "Tomato", "Lettuce", "Bread"
- "I need a device to access the internet.": "Laptop", "Desktop", "CellPhone"
- "I want a comfortable chair to sit on.": "ArmChair", "Chair", "Sofa"

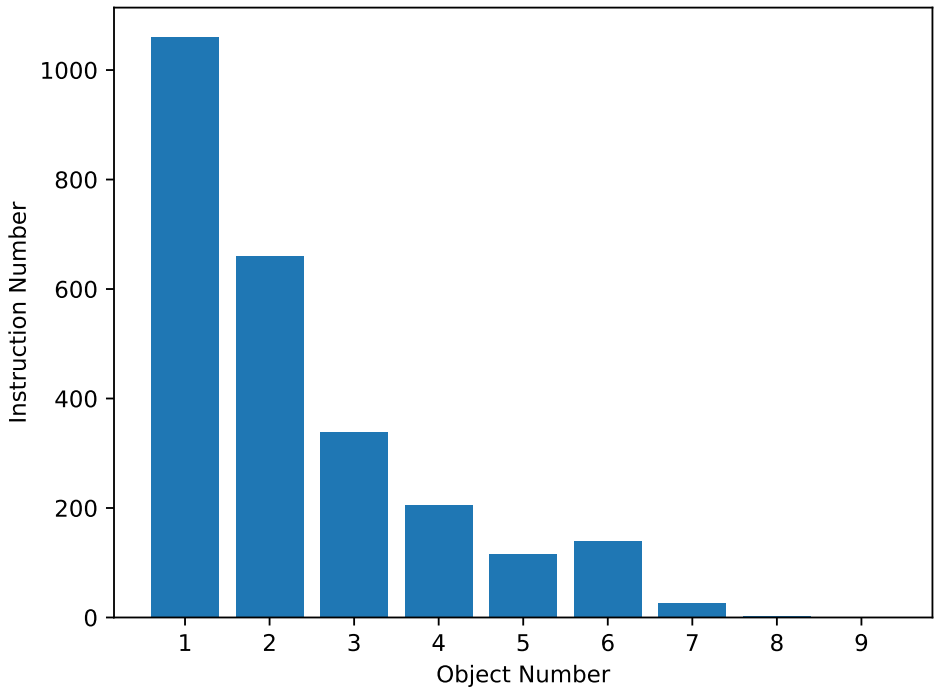

Figure 4: **Number of Objects to Satisfy for Each Instruction.** Our dataset consists of approximately 1000 instructions that can be fulfilled using only a single object, while around 600 instructions require the use of two objects. Moreover, 60% of the world-grounding mappings indicate that demand instructions require two or more objects. On average, each instruction can be satisfied by 2.3 objects.

8.1.2 Statistical Features

We conduct a statistical analysis on the world-grounding mappings we obtained, revealing an average demand instruction length of 7.5 words. Each instruction can be satisfied by an average of 2.3 objects. For more information on the number of objects needed to satisfy an instruction, please refer to Figure 4. Each object is capable of satisfying an average of 51.3 instructions. For more information on the number of instructions satisfied by each object, please refer to Figure 5. To create a word cloud, we select the 100 most frequently occurring words within the demand instructions, where a larger font size indicates higher frequency of occurrence, as shown in Figure 6. From the word cloud, we notice frequent occurrences of "I", "my", and "need", indicating that the demand instructions generated are based on user demands and not solely on the presence of objects in the environment. The word "something" also appears frequently, suggesting inference is required to determine the desired object for many demand instructions.

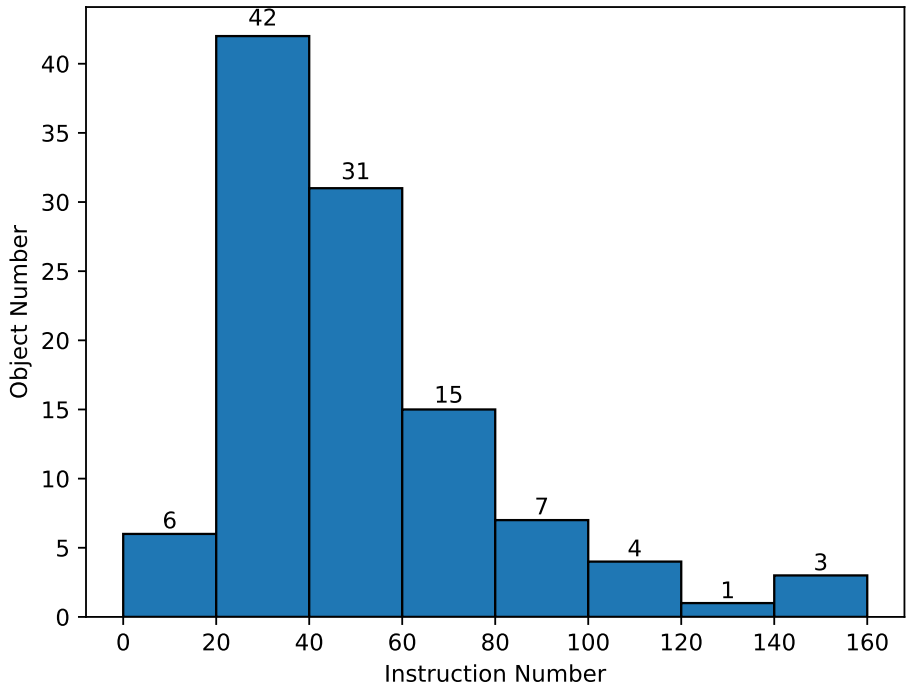

Figure 5: **Number of Instruction Satisfied by Each Object.** Our dataset indicates that there are six objects capable of fulfilling between 0 and 20 instructions, with 42 objects that can satisfy 20 to 40 instructions, and 31 objects capable of satisfying 40 to 60 instructions. On average, each object is able to fulfil 51.3 instructions.

8.2 Demand-Driven Navigation Method

8.2.1 Textual Attribute Feature Learning

Language-grounding Mappings In the case of demand instructions, there is no difference between world-grounding demand and language-grounding demand because they are both generated by GPT-3 and are independent of the environment. The key to distinguishing between world-grounding and language-grounding mappings is whether or not the objects in the mapping are environment-specific. Therefore, we do not generate language-grounding demands for attribute learning but use world-grounding demands that do not appear in the training and testing as language-grounding demands (we generate a total of about 2600 world-grounding mappings, but only 200 are selected for training and 300 for generalisability testing). For each language-grounding demand instruction, we generate ten objects that can satisfy it using GPT-3 (*i.e.*, language-grounding mappings). Here are some language-grounding mapping examples:

- "I want a comfortable chair to sit on.": " Armchair", " Recliner", " Rocking chair", " Bean bag chair", " Chaise Lounge ", " Glider chair ", " Papasan chair", " Egg chair", " Swivel chair", " Multi-position chair"
- "I need something to listen to music.": " Mobile Phone", " laptop", " Tablet", " Streaming Radio", " Portable CD Player", " Vinyl Record Player", " Portable MP3 Player", " Car Stereo", " Smartwatch", " Portable Speaker"
- "I want something to toast bread with.": " Toaster", " Toaster oven", " Convection oven", " Microwave oven", " Sandwich press", " George Foreman grill", " Panini press", " Cast iron skillet", " Electric griddle", " Hearth oven"

Figure 6: **Instruction Word Cloud.** We select the 100 most frequently occurring words in the demand instruction. The higher the word frequency, the larger the font.

- "I want something to hold a beverage.": " Water bottle", " Thermos", " Flask", " Coffee mug", " Travel mug", " Teacup", " Tumbler", " Soda Can", " Wine glass", " Beer stein"

Upon reviewing ProcThor's metadata [25], it becomes apparent that numerous language-grounding objects are omitted. This observation is inherent to language-grounding, where objects are produced by the language model and are not contingent on the environment.

t-SNE of Demand-conditioned Attribute Features To map the demand-conditioned attribute features into a 2-dimensional space, t-SNE [69] is employed, and the resulting visualisations are displayed in Fig. 7 (positive samples) and Fig. 8 (negative samples). In Fig. 7, each colour corresponds to a unique demand and its corresponding objects, with each point representing an object that can satisfy the same demand. Notably, the objects of the same colour are in close proximity to one another. Meanwhile, Fig. 8 represents the demand instruction held constant while different objects' CLIP features are combined to produce diverse demand-conditioned attribute features. The blue points signify the given demand BERT feature concatenated with objects capable of fulfilling the demand, whereas the red points denote the given demand BERT feature concatenated with objects that cannot satisfy the demand. It is evident that the blue points are centrally clustered, whereas the red points are more scattered throughout the figure.

8.2.2 Policy Learning

Pipeline in Transformer In practice, we select the k bound boxes with the highest logits in DETR and get k CLIP-visual features(512-D vector), where k is 16. These k CLIP-visual features are concatenated with the given demand BERT feature (1024-D vector), and we obtain k demand-object features (1536-D vector). The Attribute Module is then leveraged to process these k demand-object features and provide k demand-conditioned attribute features (512-D vector). Subsequently, we combine these k demand-conditioned attribute features with their corresponding DETR's bounding boxes (4-D vector) and logits (2-D vector) and introduce the resulting concatenated features to a fully connected network. Consequently, we generate k 512-D attribute-bbox-logits features.

A Transformer model is leveraged to process three input sets- the k 512-D attribute-bbox-logits features, the global visual feature, and the demand BERT feature. Specifically, we begin by feeding the k 512-D attribute-bbox-logits features into a 6-layer Transformer Encoder, generating k 512-D encoder-output features. Subsequently, we concatenate the global visual feature (1024-D vector) and the demand BERT feature (1024-D vector), and employ a fully connected layer to map this concatenation to a 512-D image-demand feature. We introduce the resultant 512-D image-demand feature as the $query$, and the k 512-D encoder-output features as the key and $value$ inputs to a Transformer Decoder. As a result, we produce a 512-D transformer-output feature.

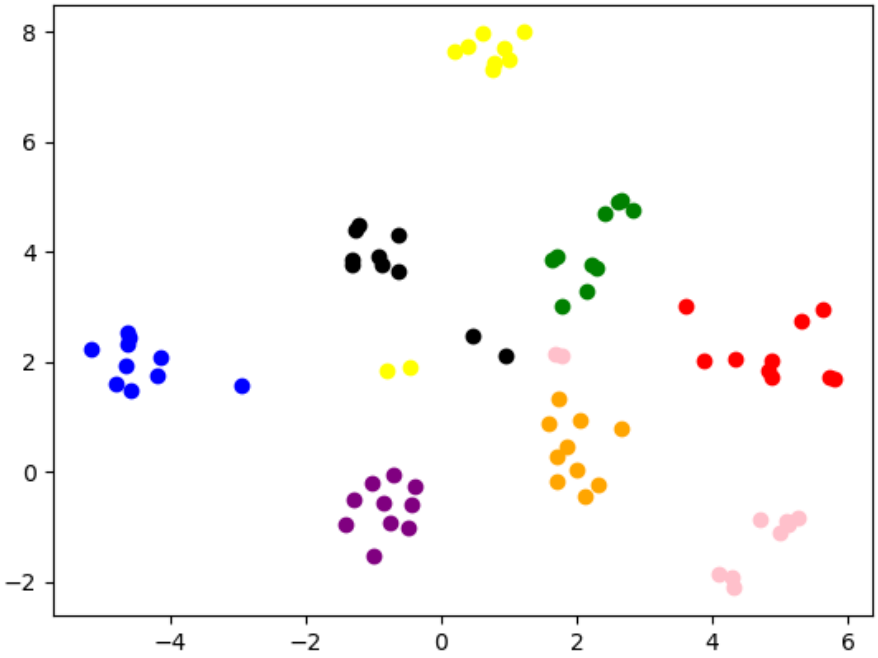

Figure 7: **t-SNE of Demand-conditioned Attribute Features (Positive Samples).** To project the demand-conditioned attribute features into a 2-D space, we utilise t-SNE. Each demand is represented by a distinct colour, with different objects capable of satisfying the same demand being represented as different points, although corresponding to the same colour. We can observe that points of the same colour cluster together, indicating that they possess similar attributes.

Concatenating the 512-D transformer-output features with the prior time-step's action embedding (16-D vector) prepares the feature set for processing by a 2-layer LSTM. The resulting LSTM-output feature (1024-D) is then leveraged to predict the action.

Trajectory Collection For every instruction in each chosen ProcThor training set room, we generate three diverse trajectories. These trajectories, which incorporate an A* algorithm [63] for planning a path approaching the object, save each RGB frame while traversing towards the target object. Additionally, other key information, such as the bounding box and initial position in the concluding step, are also recorded. In total, we acquire roughly 27,000 trajectories, with an average of 27.16 frames per trajectory for the training dataset, and roughly 3,000 trajectories, with an average of 26.18 frames per trajectory, for the validation dataset. Refer to Fig. 9 for an illustration of the trajectory length distribution.

Demand-based Visual Grounding Model We implement the Visual Grounding Model using a 6-layer Transformer Encoder. To begin, we select the k bounding boxes with the highest logits in DETR and obtain their last-layer DETR features (256-D vector), where k equals 16. We concatenate these k last-layer DETR features with their respective bounding boxes (4-D vector) and logits (2-D vector), and pass this concatenated vector through a fully connected network. The output of the network consists of k 1024-D DETR features. To obtain global image features (1024-D vector), we use a RESNET18 [64] and a 2-layer fully connected network. We then concatenate the k 1024-D DETR features, a 1024-D global image feature, a 1024-D demand BERT feature, and a 1024-D CLS token into $k + 3$ 1024-D features. We feed this concatenated vector into a Transformer Encoder. Finally, we use the CLS token for classification, where we select which of the k bounding boxes (the k bounding boxes with the highest logits selected at the beginning) is the answer.

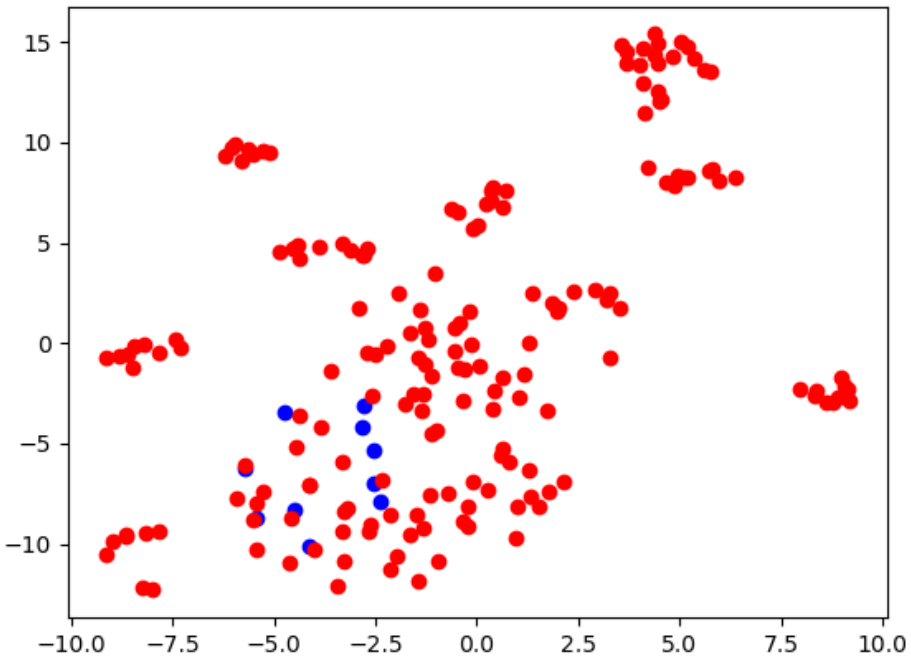

Figure 8: **t-SNE of Demand-conditioned Attribute Features (Negative Samples).** We utilise t-SNE to project the demand-conditioned attribute features into a 2-D space. To obtain different demand-conditioned attribute features, we maintain the demand instruction unchanged and concatenate multiple objects' CLIP features. Blue points represent attribute features generated by concatenating the given demand BERT feature with objects that can fulfil the demand, while red points represent attribute features generated by concatenating the given demand BERT feature with objects that cannot fulfil the demand. We can observe that the blue points gather together, while the red points are scattered throughout the entire figure.

To gather Visual Grounding training data, we randomly teleport the agent in the room and save the objects' bounding boxes when they meet the distance threshold c_{navi}. The train dataset contains approximately 1M bounding boxes and 1M images. The validation dataset contains 104K bounding boxes and 100K images.

8.2.3 Model Pre-training and Fine-tuning

Image Encoder: Vision Transformer We use the mae_vit_large_patch16 version of Vision Transformer [61, 62] as the Image Encoder to obtain global visual features. We initialise the Image Encoder with the official pre-train weights and fine-tune it using Masked-Reconstruction Loss [62] with around 1.9M images collected in the ProcThor dataset.

DETR We fine-tune the DETR model [60] using a dataset collected from ProcThor [25]. We collect the images by randomly teleporting the agent in the room and saving all the bounding boxes from the ProcThor metadata. The train dataset consists of approximately 713K images, and the validation dataset consists of 71K images. For the hyper-parameters of DETR, we set the upper limit for the number of bounding boxes at 100 and the number of object categories to 2 (one for background and one for object instances).

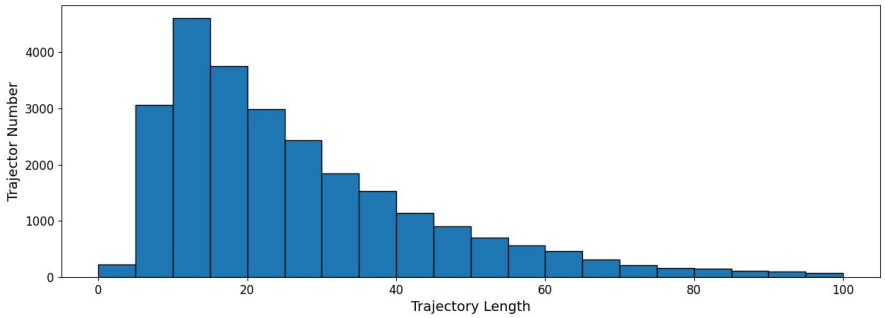

Figure 9: **The Distribution of Imitation Learning Trajectories Length.** This is the distribution of trajectory lengths used for imitation learning, with most of the trajectories clustered within 10∼30 steps.

8.3 Experiments

8.3.1 Metrics

- Navigation Success Rate (NSR): the fraction of navigation successful episodes. The threshold distance c_{navi} is 1.5m.
- Navigation Success Weighted by Path Length (NSPL) [67]: we weigh the success by the ratio of the execution path length to the shortest path length.
- Selection Success Rate (SSR): the fraction of selection successful episodes. The threshold IoU c_{sele} is 0.5.

8.3.2 Baselines

All baselines involving reinforcement learning (RL) algorithms use PPO [70]. For further information about the training pipeline of VTN [6] and ZSON [11], please refer to their respective papers and official codebases. All baselines are trained on a machine equipped with A100 40G*2, 128-core CPU, 192G RAM. We train each baseline for 7 days, which is around 1.8M steps in reinforcement learning. VTN's pre-imitation learning takes 3 days, approximately 30 epochs before using the best model obtained on the validation set to perform RL fine-tuning.

VTN-object This baseline involves the VON task whose trained model will be used in VTN-GPT. To perform the baseline, we use the original training pipeline of the VTN, except for a modification of the target object categories to 109 classes (*i.e.*, all the object categories in the DDN). For pre-imitation learning of the VTN, we use the same trajectory dataset as our method. The VTN model is then fine-tuned using PPO.

VTN-demand We replace the vector that indicates the target object with the demand BERT feature while keeping the rest of the pipeline unchanged.

VTN-CLIP-demand We replace the vector that indicates the target object with the demand BERT feature and DETR features used in VTN with CLIP features while keeping other parts unchanged.

VTN-GPT* For this baseline, we select the best-performing model in VTN-object (as determined by the best success rate in the validation set) and combine it with GPT-3 for testing. Since VON restricts the range of target objects, we utilise scene metadata, such as the categories of objects in the scene. Concretely, we first encode the 109 object category names in ProcThor with CLIP-text Encoder and obtain the corresponding CLIP-textual feature set F_{target}. At the start of each navigation evaluation, GPT-3 generates a language-grounding object o_l that satisfies the given demand instruction. Next, we use the CLIP-text Encoder to encode o_l, producing the corresponding CLIP-textual feature, f_l. Cosine similarity is computed between f_l and all features in F_{target}, and the object corresponding to the most similar feature is selected as the target object provided to the chosen VTN-object model.

When the agent performs Done, if the step limit is not yet reached and the target object is not yet found, GPT-3 will regenerate a different language-grounding object, and then repeat the above process to select the target object, until the object satisfying the demand is found (success) or the step limit is reached (failure).

ZSON-object We train the ZSON-object model using its original pipeline whose trained model will be used in ZSON-GPT. We first train a ImageGoal ZSON model. We use the images that have objects present at a distance less than c_{navi} as target images (use CLIP-image Encoder to obtain CLIP-visual features as target inputs), and then ask the agent to find the objects in the images. When testing, we use the category name of objects (109 object categories in total) as the target object (use CLIP-text Encoder to obtain CLIP-textual features as target inputs) and ask agent to find the objects.

ZSON-demand We use the best model in ZSON-object and then use the CLIP-textual feature of the demand instruction as the target inputs for the agent to find the object that satisfies the demand.

ZSON-GPT Similar to VTN-GPT, we use the best model in ZSON-object for testing. At the beginning of each evaluating navigation, GPT-3 generates a language-grounding object o_l that can be used to satisfy the given demand instruction. We then use the CLIP-textual feature of the language-grounding object f_l as target inputs and ask the agent to find the language-grounding object. When the agent performs Done, if the step limit is not yet reached and the target object is not yet found, GPT-3 will regenerate a different language-grounding object, and then repeat the above process to select the target object, until the object satisfying the demand is found (success) or the step limit is reached (failure).

CLIP-Nav-GPT/MiniGPT-4 Since our demand instructions do not instruct the agent to act step-by-step like the instructions in VLNs, there is also no need for LLM to decompose the instructions. We use the CLIP model to indicate directions as in CLIP-Nav and use GPT-3 or MiniGPT-4 to identify whether the objects in the images satisfy the demand at each time step.

FBE-GPT/MiniGPT-4 We use a heuristic search algorithm, FBE, as an exploration policy. FBE builds occupancy maps and exploration maps, selecting the nearest unexplored node each time. We use GPT-3 or MiniGPT-4 to identify whether the objects in the images satisfy the demand at each time step.

GPT-3+Prompts* We use Prompt Engineering and query GPT-3 what action should be chosen to perform. The prompts are shown below:

Imagine you are a home robot and your user will give you a instruction, and you need to find objects that can satisfy the needs in the instruction. I will tell you the objects you can currently see and the corresponding distances.
Your task success criteria is that the object that meets the user's demand appears in your field of view and is less than 1.5m away from you. You should find this object as soon as possible.
The actions you can select are listed below, and the effect after execution is shown after the colon. For each step you can only choose one of the following six actions.
RotateRight: Rotate right by 30 degrees
RotateLeft: Rotate left by 30 degrees
LookUp: Look up by 30 degrees
LookDown: Look down by 30 degrees
MoveAhead: Move forward by 0.25m
Done: End the task
When you select Done, you also need to output the name of the selected object in your field of view for the user's needs and use a space to separate Done from the name of the object.
This is the user's instruction to you: $instruction$
The objects you can currently see are:
name: $object_1$, bounding box: $Bbox$, distance: d.
name: $object_2$, bounding box: $Bbox$, distance: d.
....

Note: The bounding box is the upper left and lower right coordinates of the object in the image, and the distance is the distance between the object and the robot.
You may reason about the user's needs based on the objects you can see.
Your previous actions are:$action_1$, $action_2$, $action_3$....
Please select only an action in [RotateRight, RotateLeft, LookUp, LookDown, MoveAhead, Done].

In the above prompts, "...." represents other objects not listed. During the testing, the name, bounding box, and distance of the objects are obtained from the metadata of the environment.

Table 2: VON results.

	Seen Scene		Unseen Scene	
Method	SR	SPL	SR	SPL
Random-object	2.3	2.3	2.3	2.3
ZSON-object	4.4(0.3)	2.9(0.3)	2.1(0.3)	1.9(0.4)
VTN-object	4.9(1.1)	4.9(1.1)	3.6(1.2)	3.3(1.0)

MiniGPT-4 The prompts are shown below:

Give the following image: $image$. You will be able to see the image once I provide it to you. Please answer my questions.
Imagine you are a home robot and your user will give you a instruction, and you need to find objects that can satisfy the needs in the instruction.
Your task success criteria is that the object that meets the user's demand appears in your field of view and is less than 1.5m away from you. You should find this object as soon as possible.
The actions you can select are listed below, and the effect after execution is shown after the colon.
For each step you can only choose one of the following six actions.
RotateRight: Rotate right by 30 degrees
RotateLeft: Rotate left by 30 degrees
LookUp: Look up by 30 degrees
LookDown: Look down by 30 degrees
MoveAhead: Move forward by 0.25m
Done: End the task
When you select Done, you also need to output the name of the selected object in your field of view for the user's needs and use a space to separate Done from the name of the object.
In the image, you can see these objects:
$object_1$, $object_2$, $object_3$, $object_4$...
This is the user's instruction to you: $instruction$
You may reason about the user's needs based on the objects you can see.
Your previous actions are: $action_1$, $action_2$, $action_3$....
Please select only an action in [RotateRight, RotateLeft, LookUp, LookDown, MoveAhead, Done].

8.3.3 Additional Results

Since VTN-GPT and ZSON-GPT require a VON model as a navigation model, we train the two models, VTN-object and ZSON-object, using their official code, and the results are shown in Tab 8.3.2.

8.4 Ethics

Based on the information we currently have, we have not identified any moral or ethical issues.