Open Scene Graphs for Open-World Object-Goal Navigation

Joel Loo[1*], Zhanxin Wu[1*], and David Hsu[1]

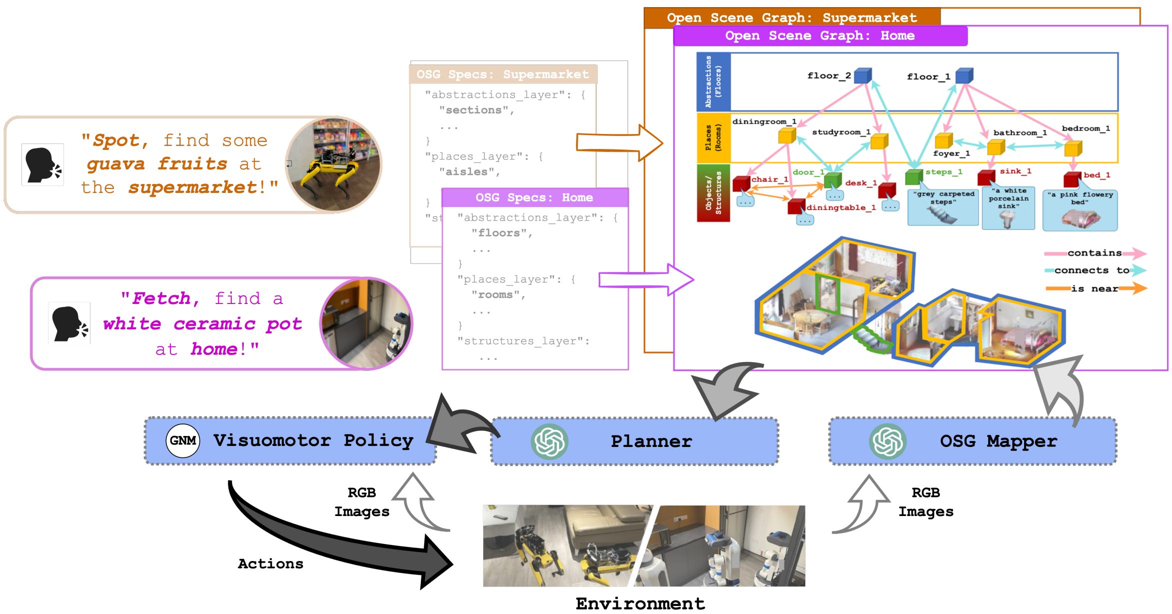

Fig. 1. *Explorer* addresses the problem of *open-world object-goal navigation*: *i.e.* building a system for navigating to an open-vocabulary object goal, that generalises across embodiments and environments. *Explorer* is a robot system fully made up of foundation models to achieve such generalisation. To compose these models effectively, we propose the *Open Scene Graph* (OSG) to act as a persistent scene memory for the models. The OSG is a hierarchical, topo-semantic scene representation, whose structure can be configured to appropriately represent different environments.

***Abstract*—Can we build a system to perform object-goal navigation (ObjectNav) and other complex semantic navigation tasks in the open world? Composing LLMs that are strong semantic reasoners with robotics foundation models that generalise across environments and embodiments seems a viable avenue. While selecting the right representations to connect them effectively is crucial, existing works often prompt LLMs with scene representations that are uninformative, unstructured and constructed with methods that generalise poorly. To address this gap, we propose the *Open Scene Graph* (OSG), a rich, structured topo-semantic representation, along with an *OSG mapper* module composed fully from foundation models. We demonstrate that OSGs facilitate reasoning with LLMs, enabling a greedy LLM planner to outperform existing LLM approaches by a wide margin on ObjectNav benchmarks in diverse indoor environments. We take a step towards an open-world ObjectNav system by building the fully foundation model-based Explorer system from an LLM planner and a generalisable visuomotor policy, with OSGs built online by our mapper to connect them. We show that Explorer is capable of effective object-goal navigation in the real world across different robots and novel instructions.**

I. Introduction

Consider being asked to find a bottle of wine in an unfamiliar house. We might first reason about its possible location: it might be kept in the kitchen or a wine cellar. We would then guess where the kitchen or wine cellar might be and find our way there. This *object-goal navigation* (ObjectNav) task is an instance of the broader domain of semantic navigation: navigation tasks that go beyond geometric scene understanding, vitally requiring semantic reasoning abilities and commonsense priors about human environments. Formally, ObjectNav requires an agent to navigate to a specified object category in a novel, unmapped scene [16]. Our ability to perform such semantic navigation tasks also transfers to new environments - *e.g.* if we were in a restaurant or hotel, and new embodiments - *e.g.* if we were lugging a heavy shopping trolley or in a wheelchair. In this work, we consider the problem of *open-world object-goal navigation* (ObjectNav) that can *generalise across environments and embodiments*. While recent ObjectNav methods exhibit strong performance, broad generalisation remains an issue [10]. Large Language Models' (LLMs) semantic reasoning and rich open-world knowledge hold promise for enabling capable and generalisable ObjectNav [18, 13]. By composing them with recent robotics foundation models for perception [6, 8] and control [14, 15] that transfer over embodiments [5], we can build systems capable of ObjectNav across environments and embodiments.

However, effectively harnessing these models' abilities to

*Equal contribution.
[1]School of Computing, National University of Singapore

perform robot system functions like planning and localisation requires an appropriate scene representation to retain and organise the information needed by the models. Existing works using LLMs often employ unstructured, uninformative representations that do not capture the scene's structure and contents in an organised way [4, 18, 13], limiting LLMs' ability to reason. Others recognise the need for rich, structured representations like scene graphs [11, 9], but rely on metric SLAM pipelines with limited generalisation across environments and embodiments [1]. Thus to enable strong reasoning and to generalise, the representation should be structured and mainly reliant on semantic information.

In this work we focus on establishing the feasibility of building a single robot system for effective open-world ObjectNav, by composing it fully from foundation models. The crux of this approach lies in designing a rich structured scene representation that facilitates LLM reasoning, along with an associated mapping pipeline that promotes generalisation. Specifically, we propose the Open Scene Graph (OSG), a hierarchical, topo-semantic, open-vocabulary representation of a scene's contents and topology. It has a configurable structure that can be adapted by the user to appropriately represent various environment types. OSGs are built online with our OSG mapper, a mapping pipeline fully composed from VFMs and LLMs. We construct Explorer, our open-world ObjectNav system, from the OSG mapper, an LLM-based planner and a General Navigation Model visuomotor policy, with the OSG used as the central scene memory. Through experiments in simulation, and also in the real world across two different robots, we show that Explorer enables effective open-vocabulary ObjectNav that generalises zero-shot across diverse indoor environments and robots. In particular, we find that an LLM-based system shows improved reasoning for ObjectNav when using OSGs, enabling it to outperform existing LLM-based approaches on standard benchmarks by a wide margin.

II. Problem Formulation

In the ObjectNav task, a robot is assigned to find an instance of a specified object category in a novel indoor environment with no prior map. We allow for different robot embodiments so long as they take linear/angular velocity commands and have a forward-facing RGB camera, without restricting the camera type or its mounting. We do not assume metric information like pose estimates is available. This setting encourages solutions that generalise over environments and embodiments.

III. Approach

A. Overview

We propose *Explorer*, a modular robot system for open-world ObjectNav, comprising an OSG acting as a structured scene memory, along with various subsystems composed from foundation models. The 3 subsystems in *Explorer* are: **(1)** an *OSG mapper* that estimates the robot's state and builds the OSG online (Section III-C); **(2)** a *planner* that searches over the OSG to propose regions to explore that are potentially near the target object, and generates plans to reach them (Section III-D); **(3)** and an *image-goal visuomotor policy* that navigates the robot to the next waypoint in a given plan (Section III-E).

Each subsystem in *Explorer* is composed from one or more foundation models, drawn from the following 4 types of models: a Large Language Model (LLM), a General Navigation Model (GNM), a Visual Foundation Model (VFM) for Visual Question-Answering (VQA), and a VFM for open-set object detection. In particular, *Explorer* uses: OpenAI's GPT-3.5 (LLM); ViNT [14] (GNM); BLIP-2 [7] (VQA); GroundingDINO [8] (object detector).

B. Open Scene Graphs

An *Open Scene Graph* (OSG) is a layered, directed graph that represents a scene's contents and spatial structure at multiple levels of abstraction. Formally it is a heterogeneous, simple directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ organised into N layers. It serves to **(1)** contain image/textual information needed for planning, localisation and control with foundation models, and to **(2)** organise the information in a semantically meaningful way with multiple levels of abstraction to improve the efficiency and efficacy of decision-making, particularly with LLM-based planners [11, 19]. It is also **(3)** designed to be built purely with foundation models, fully harnessing their open-set abilities.

We first give a high-level description of the node and edge types in an OSG, and then give a formal description of these elements to establish layers needed in OSG.

Nodes. The nodes of an OSG represent information about a scene's contents (i.e. Objects and Structures) and the semantically meaningful spatial regions (i.e. Places and Region Abstractions) within it. **(1) Objects**. We define objects as static scene elements occupying a spatially localised region. They enable a rich, functional understanding of the scene. Objects can also serve as effective landmarks. In particular, we recognise places and objects using their object features - i.e. an aggregated list of nearby objects and their associated textual descriptions. Finally, objects can also serve as goals for navigation. **(2) Places.** Places are the smallest semantically meaningful spatial regions with a particular functional and semantic meaning in a scene - *e.g. study-rooms* for work - which is information valuable for semantic reasoning. Also, we define the robot's state in our system as a Place node in the OSG. **(3) Region Abstractions.** These organise the set of spatial regions by partitioning them into spatially and semantically close subsets. In the example, *rooms* are organised with the *floor* abstraction, forming two separate clusters of connected rooms based on their elevation. Such hierarchical abstractions can enable hierarchical planning that scales to expansive environments. **(4) Structures.** Structures are spatially localised *building elements* that physically connect different spatial regions, *e.g.* doors that connect rooms. Structures help capture information on reachability and spatial connectivity in a scene, which is valuable for planning.

Edges. We represent spatial and semantic relationships between scene elements via 3 types of directed edges. The relationships and corresponding edge types are: **(1) Spatial**

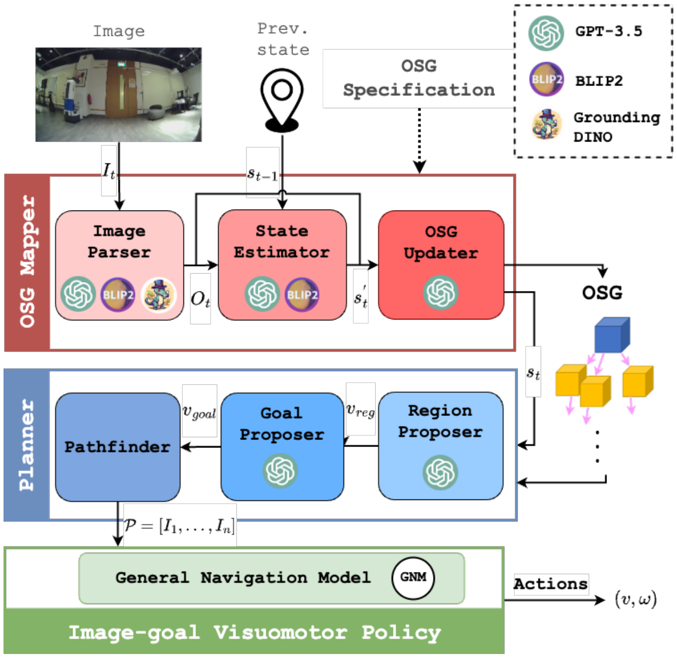

Fig. 3. System architecture of *Explorer*. The OSG mapper takes as input the current RGB observation and robot's previous state (Place node). It estimates the current state and updates the OSG, guided by the OSG specification. The planner reasons over the OSG to propose a goal node to explore. It then finds a reachable path $\mathcal{P}$, consisting of a sequence of images of the nodes along the path. The image-goal visuomotor policy is tasked with navigating to each node in turn, conditioned on an image of the next node along the path.

connectivity. "`connects to`" indicates that the destination node is reachable from the source node. These specify the topological structure of the environment. **(2) Spatial proximity.** "`is near`" indicates spatial proximity between nodes, and is particularly used to determine nearby objects that can be used as object features. **(3) Hierarchy.** "`contains`" specifies the hierarchy of spatial abstractions and captures the concept of spatial containment. These edges define a tree over the nodes of the OSG.

Layers. An OSG is made up of layers, each containing nodes of a distinct type. **(L1)** The lowest layer contains *Object* nodes as leaf nodes, each of which represents a distinct object and has an associated textual description and image. **(L2)** This contains *Structure* nodes, that are also leaf nodes. In addition, they can be distinguished from Objects since they also capture spatial connectivity. **(L3)** This contains *Place* nodes, which are the finest-resolution spatial regions that can contain the robot - *e.g.* a room in a home. **(L4-N)** The kth layer, for $k \geq 4$, is a configurable *Region Abstraction* that partitions and organises the spatial regions in the $(k-1)$th layer - *e.g.* rooms (Places) can be clustered into floors (Region Abstraction). The OSG specification only requires that Layers 1 and 3 be specified to facilitate planning and localisation, with Layer 2 and Layers 4 onward being optional.

C. OSG mapper

The OSG mapper builds OSGs online, structuring them in accordance with OSG specifications given by the user at the start of the navigation episodes. At a high level, it processes RGB observations into textual information, which is used in a structured set of queries to an LLM to estimate the robot's state and update the OSG. Specifically, it uses an LLM, a VQA model and open-set object detector to do so. The OSG mapper is a sequential pipeline comprising 3 modules: **(1)** an image parser to extract semantic information from RGB observations; **(2)** a state estimator; **(3)** an OSG updater which updates the OSG using the semantic information and predicted state.

1) Image parser: The image parser extracts the semantic information O_{det} needed for planning, localisation and control from RGB images, using the VQA and object detector VFMs. It identifies and lists the objects in an image, each associated with a rich textual description and image crop. It also identifies the Place the robot is currently in, by querying the VQA model to describe its current location. Examples in Appendix E.

2) State estimator: The state estimator uses O_t to predict the robot's current state in the OSG, *i.e.* the Place node it is in. If the state estimator determines the robot is in a novel location, it triggers the OSG updater to add a new Place node into the OSG. Given the label for the robot's current location and the object features from the observation, the state estimator queries the LLM to identify nodes in the OSG with semantically similar labels by comparing their object features.

3) OSG updater: The OSG updater integrates information from incoming observations into the OSG. To do so, it first updates nodes and edges in Layers 1-3 based on the observations by prompting the LLM to compare their object features with those stored in the OSG. The contents of abstract, higher layers may not be directly observable from egocentric images, and are inferred from the updated Objects, Structures and Places in the OSG. Further details are given in Appendix E.

D. Integrating an LLM planner into Explorer

The planner semantically reasons about possible locations for the target object using the OSG, then plans paths to search these promising areas. Possible target object locations are suggested using the **(1)** Region Proposer and **(2)** Goal Proposer modules. The Region Proposer guides the LLM to reason hierarchically about *spatial regions*, and propose a Place node close to or containing the target object. Given a proposed Place node, the Goal Proposer queries the LLM to select an Object/Structure leaf node contained in this Place as a concrete goal for the robot to navigate to. Then, the planner's **(3)** Pathfinder module searches for a feasible path in the OSG using Dijkstra, and successively issues waypoints on the path to the visuomotor policy as the robot navigates. Pathfinder tracks the robot's state to determine the next waypoint node, and uses the image crop of it stored in the OSG to command the visuomotor policy.

E. Integrating a visuomotor policy into Explorer

To navigate according to the planned paths in the real world, *Explorer* uses an image-goal visuomotor policy. Recent work on such policies [14, 15] highlight that they are effective substrates on which to build cross-embodiment foundation models for navigation with. In particular, *Explorer* uses the

ViNT GNM zero-shot, by commanding with an image crop of the next subgoal node in the planned path. These crops are stored in the OSG by the OSG updater.

IV. Experiments

Our experiments answer the following questions: **(1)** Is LLM planning with OSGs effective at object-goal navigation over diverse environments? **(2)** How well does *Explorer* perform in the real world, across different robot embodiments?

In these tests, *Explorer* explores unmapped areas, continuously updating a scene memory in the form of a OSG as it does so. Using this OSG, it localises and plans about where to explore next to find the target object category.

A. Experimental setup in simulation

Our simulation experiments evaluate the utility of OSGs for ObjectNav, especially compared to LLM-based approaches.

Datasets and metrics. We evaluate our approach on the full Gibson validation set and 400 randomly sampled episodes from HM3D-Semantics v0.1 validation set, that are equally split among all scenes in the dataset. We follow the Habitat ObjectNav Challenge setup [16] and use the standard metrics of success rate, success-weighted path length (SPL) and distance-to-goal (DTG).

Baselines. To maintain consistency in controllers across baselines, we use a Fast Marching Method [12] (FMM) controller. For *Explorer* variants, we use RGB-D information in the FMM controller to enable it to navigate to image goals. Further details can be found in Appendix G. The evaluated baselines and variants are:

a) *Greedy LLM*: Similar to [4]. Scene information is represented to LLM as a list of objects detected in the current timestep's image input. Using this, LLM greedily selects an object to explore towards. Uses ground-truth object annotations.
b) *Language Frontier Guide* (LFG) [13]: Maintains a metric frontier map that accumulates object information. Scene information is represented to LLM as a list of object clusters extracted from frontiers in the metric map. LLM scores the frontiers, and acts as a search heuristic. Uses ground-truth object annotations.
c) *Explorer-FMM-GT*: Variant of our approach that uses ground-truth object annotations.
d) *Explorer-FMM*: Variant of our approach that does not use any ground-truth information.

B. Experimental setup in real-world

Our real world experiments evaluate the ability of *Explorer* to perform zero-shot object-goal navigation with open-vocabulary instructions, across different robot embodiments. We conduct tests in an open-plan apartment environment comprising a living room, dining room and kitchen.

Robots. We deploy *Explorer* on two real-world robot systems: (1) Spot: A quadrupedal robot, with a 170° fisheye RGB camera mounted ~ 0.8m above ground and (2) Fetch: A differential-drive mobile manipulator, with a 79° Realsense RGB camera mounted ~ 1.4m above ground.

TABLE I
Comparison against LLM-based object-goal navigation approaches on HM3D validation set

Method	Success (↑)	SPL (↑)	DTG (↓)
Greedy LLM	0.275	0.080	5.078
LFG [13]	0.675	**0.389**	2.411
Explorer-FMM-GT	**0.775**	0.380	**1.702**
Explorer-FMM	0.693	0.283	2.338

TABLE II
Real-world success rate of *Explorer* across different robots

Object goal	*Spot*	*Fetch*
`guitar`	4/5	4/5
`dish washer`	5/5	4/5

C. Effectiveness on ObjectNav over diverse environments

We observe that *Explorer* shows strong zero-shot performance on ObjectNav across diverse indoor environments. We attain a high success rate on HM3D, which contains a diverse range of household scenes. We also outperform recent methods using LLMs by a wide margin (Table I). We find that rich, structured representations that encode not just object information but also the environment's structure and layout can enhance exploration. From Table I, LFG improves significantly over Greedy LLM by providing richer and more comprehensive information (*i.e.* listing objects from accumulated metric map compared to listing objects within field-of-view) that is better structured (*i.e.* objects organised into spatial clusters compared to a flat list). OSG-based LLM planning improves further over LFG, which we attribute to OSGs being richer and more informative: by organising scene elements with semantically meaningful abstraction, it provides crucial semantic structure that improves LLM planning and hence ObjectNav performance. In general, while our topo-semantic approach does not let us optimise for shortest geometric paths, the *Explorer* variants still remain competitive with state-of-the-art baselines in SPL and DTG. More results in Appendix H.

D. How well does Explorer perform in the real world?

Explorer is able to perform object-goal navigation across embodiments, with open-vocabulary instructions in the real world. Our tests search for 2 uncommon object types: "guitar" and "dish washer". Table II highlights that *Explorer* succeeds often, with similar performance across both Spot and Fetch.

V. Discussion

We present *Explorer*, a fully foundation model-based system for *ObjectNav* that generalizes across new instructions, environments, and embodiments. We find that it performs strongly on simulation benchmarks and when deployed zero-shot across different robots in the real world with novel instructions. This is enabled by our *Open Scene Graph*, which captures and organises rich information about the scene's contents and structure, and our foundation model-based OSG mapper for building OSGs.

REFERENCES

[1] César Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian D. Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32:1309–1332, 2016.

[2] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *In Neural Information Processing Systems (NeurIPS)*, 2020.

[3] Junting Chen, Guohao Li, Suryansh Kumar, Bernard Ghanem, and Fisher Yu. How To Not Train Your Dragon: Training-free Embodied Object Goal Navigation with Semantic Frontiers. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

[4] Vishnu Sashank Dorbala, James F. Mullen, and Dinesh Manocha. Can an embodied agent find your "cat-shaped mug"? llm-based zero-shot object navigation. *ArXiv*, abs/2303.03480, 2023.

[5] Yafei Hu, Quanting Xie, Vidhi Jain, Jonathan Francis, Jay Patrikar, Nikhil Keetha, Seungchan Kim, Yaqi Xie, Tianyi Zhang, Shibo Zhao, Yu Quan Chong, Chen Wang, Katia Sycara, Matthew Johnson-Roberson, Dhruv Batra, Xiaolong Wang, Sebastian Scherer, Zsolt Kira, Fei Xia, and Yonatan Bisk. Toward general-purpose robots via foundation models: A survey and meta-analysis, 2023.

[6] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[7] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023.

[8] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[9] Abhinav Rajvanshi, Karan Sikka, Xiao Lin, Bhoram Lee, Han-Pang Chiu, and Alvaro Velasquez. Saynav: Grounding large language models for dynamic planning to navigation in new environments, 2023.

[10] Santhosh K. Ramakrishnan, Devendra Singh Chaplot, Ziad Al-Halah, Jitendra Malik, and Kristen Grauman. Poni: Potential functions for objectgoal navigation with interaction-free learning. In *Computer Vision and Pattern Recognition (CVPR), 2022 IEEE Conference on*. IEEE, 2022.

[11] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. In *7th Annual Conference on Robot Learning*, 2023.

[12] J A Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[13] Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *7th Annual Conference on Robot Learning*, 2023.

[14] Dhruv Shah, Ajay Sridhar, Nitish Dashora, Kyle Stachowicz, Kevin Black, Noriaki Hirose, and Sergey Levine. ViNT: A foundation model for visual navigation. In *7th Annual Conference on Robot Learning*, 2023.

[15] Ajay Sridhar, Dhruv Shah, Catherine Glossop, and Sergey Levine. NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration. *arXiv pre-print*, 2023.

[16] Karmesh Yadav, Santhosh Kumar Ramakrishnan, John Turner, Aaron Gokaslan, Oleksandr Maksymets, Rishabh Jain, Ram Ramrakhya, Angel X Chang, Alexander Clegg, Manolis Savva, Eric Undersander, Devendra Singh Chaplot, and Dhruv Batra. Habitat challenge 2022. https://aihabitat.org/challenge/2022/, 2022.

[17] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997.

[18] Bangguo Yu, Hamidreza Kasaei, and Ming Cao. L3mvn: Leveraging large language models for visual target navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3554–3560, 10 2023.

[19] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. Step-back prompting enables reasoning via abstraction in large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Appendix

A. System details

TABLE III
HYPERPARAMETERS USED IN THE *Explorer* SYSTEM

Subsystem	Hyperparameter	Value
OSG mapper	Centroid nearness, β_{pix}	100 (pixels)
	Bounding box overlap, β_{IOU}	0.1 (IoU)
	Min. object size threshold	200 (pixels^2)
	LLM temperature	0.3
Planner	LLM temperature	0.3
Visuomotor policy (FMM)	Occupancy map resolution	0.05 (m)
	Occupancy map size	4.8 (m)
	Min. obstacle height	0.5 (m)
	Turning action angle	30 (deg)
	Forward action distance	0.25 (m)
Visuomotor policy (GNM)	Stopping threshold	2.25
	Max linear velocity	0.5 (m/s)
	Max angular velocity	0.3 (rad/s)

1) Explorer system hyperparameters: Table III highlights the hyperparameters used in each subsystem of *Explorer*. The OSG mapper's and planner's hyperparameters are kept the same across both simulation and real-world tests, and across robot platforms in the real world. Some of the thresholds are sensor-dependent and can benefit from tuning for a specific sensor - *i.e.* centroid nearness and minimum object size thresholds that are expressed in pixel-based units, and hence depend on image resolution or field-of-view. In practice, our centroid nearness threshold is defined to be large enough that sufficient object features can be accumulated when using both RGB sensors in simulation and on our robots in the real world.

2) Statistics of queries for foundation models: Table IV provides the number of queries for each foundation model. In each episode, BLIP2 is prompted to predict the label of the current Place node and the observed object attributes, including color and material. Additionally, GroundingDINO provides object detection, including object labels and their corresponding bounding boxes in the observations. The LLM is responsible for parsing this textual information from BLIP2 and GroundingDINO to construct and update the OSG and plan based on the OSG.

3) Statistics of nodes and edges in OSG: Table V provides the average number of nodes and edges in OSG constructed by Explorer-FMM-GT in HM3D environments. In HM3D environments, there are typically 2 or 3 floors, with an average of 7.350 rooms per floor, and the maximum reaches 19 rooms per floor. Table V demonstrates that, on average, the system searches through 3 rooms to reach the goal. We showcase the efficiency of Explorer-FMM-GT in finding the goal object using commonsense priors derived from foundation models.

B. OSG specification

1) OSG abstract specification: We provide an example of an OSG specification described in **??**, formatted as a JSON string. For brevity, this specification contains a single region abstraction layer. The OSG mapper takes this string as input and uses it to construct OSGs. We later show examples of how this template specification can be specialised for particular environments, namely household and supermarket scenes.

```
{
   "AbstractionType": {
      "layer_type": "Region Abstraction",
      "contains": ["PlaceType"],
      "connects to": ["StructureType"]
   },
   "PlaceType": {
      "layer_type": "Place",
      "contains": ["Object"],
      "connects to": ["PlaceType", "
         StructureType"]
   },
   "StructureType": {
      "layer_type": "Structure",
      "is near": ["Object"],
      "connects to": ["PlaceType", "
         AbstractionType"]
   },
   "Object": {},
   "State": ["PlaceType"]
}
```

2) OSG specification example for household environments: We provide the OSG specification used for ObjectNav tasks in household environments in our HM3D and Gibson simulation tests. "Rooms" are often the clearest and most fine-grained division of space in homes, and are usually connected by traversable structures in the form of "stairs" and "entrances". In multi-storey homes, an additional "floors" layer further organises rooms according to the level they belong to.

```
{
   "floor": {
      "layer_type": "Region Abstraction",
      "contains": ["room"],
      "connects to": ["stairs"]
   },
   "room": {
      "layer_type": "Place",
      "contains": ["object"],
      "connects to": ["entrance", "room",
          "stairs"]
   },
   "stairs": {
      "layer_type": "Structure",
      "is near": ["object"],
      "connects to": ["floor", "room"]
   },
   "entrance": {
      "layer_type": "Structure",
      "is near": ["object"],
      "connects to": ["room"]
```

TABLE IV
QUERY TIMES FOR FOUNDATION MODELS USED IN THE EXPLORER-FMM SYSTEM

Model	Function	Avg. no. of queries per episode HM3D	Gibson
VQA (BLIP2)	*Image parser*	67.544	84.936
Object Detector (GroundingDINO)	*Image parser*	7.836	7.592
LLM (OpenAI GPT-3.5)	*State estimator*	58.718	45.804
	OSG updater	41.091	33.094
	Planner	50.739	45.455

TABLE V
AVERAGE NUMBER OF NODES AND EDGES IN EXPLORER-FMM-GT

Nodes			Edges		
Places	**Structures**	**Objects**	"contains"	"connects to"	"is near"
3.030	8.476	81.253	84.283	18.915	18.729

```
    },
    "object": {},
    "state": ["room"]
}
```

3) OSG specification example for supermarkets: We provide the OSG specification used for our mapping tests with Gibson supermarket scenes. Supermarkets are largely divided into distinct aisles that can be directly connected to each other - thus, we only specify "aisles" as Places and do not specify Structures. Due to the small size of the Gibson supermarket scene, we do not consider higher-level region abstractions, though we note that in larger supermarkets and department stores, "sections" or "floors" might be reasonable region abstractions.

```
{
    "aisle": {
        "layer_type": "Place",
        "contains": ["object"],
        "connects to": ["aisle"]
    },
    "object": {
    },
    "state": ["aisle"]
}
```

4) OSG specification example for household environments in real-world experiments: We provide the OSG specification used for testing *Explorer* in real world.

```
{
    "room": {
        "contains": ["object"],
        "connects to": ["entrance", "room"]
    },
    "entrance": {
        "is near": ["object"],
        "connects to": ["room"]
    },
    "object": {},
    "state": ["room"]
}
```

C. OSG mapper: Image parser module

We prompt BLIP-2 to get the instance label v^p_{label} for the current place the agent is observing.

```
What <PlaceType> are we in?
```

While our specification allows for multiple Place types v^p_{cls}, our system currently assumes that only one Place type (*e.g.* rooms or aisles) per environment for simplicity, and directly queries BLIP-2 for the Place label without first identifying the type of Place it is in. We note that this can potentially be extended to settings with multiple Place types by first querying BLIP-2 to identify which specified Place type is closest to the observations.

For detailed information about objects and structures, we query BLIP-2 to obtain their material and color.

```
What color is the <Object>/<StructureType>
?
What material is the <Object>/<
StructureType> made of?
```

D. OSG mapper: State estimator module

We prompt the LLM to determine whether two places are the same location based on their object features. This is used in the state estimator when the current location has the same BLIP-2 issued label as nodes already in the OSG. In this case, the state estimator attempts to determine if the current location has already been seen before, and if so identify the specific visited node it matches.

Our prompt encourages the LLM to reason about the similarity of composition of observed objects and structures in both places, and about the similarity of the spatial relationships between objects/structures in each scene. It also directs the LLM to focus on larger objects, which are more robustly and consistently detected than smaller objects. We provide several examples of such reasoning with few-shot prompting to guide the LLM.

```
There are two descriptions of what I observe from two viewpoints. Please assess the shared objects and spatial relationship in the descriptions to determine whether these two positions are in the same <PlaceType>. Please reply True or False. Always follow the format: Reasoning: <your reasoning>. Answer: <your answer>

Description 1: On the left, I can see a brown wood headboard, a white paper pillow. On the right, I can see a black metal television, gray plastic laundry basket, white wood closet dresser, and brown wood drawer. In front of me, I can see a white wood bed, white wood window, brown metal lamp, brown wood dresser, brown wood dresser nightstand, black silk curtain, white plastic curtain, white metal wall lamp, brown wood drawer. Behind me, I can see a brown wood cabinet.

Description 2: On the left, I can see a white wood door. On the right, I can see a white wood bed, white glass lamp, white glass window, white plastic curtain, brown wood dresser nightstand, white glass window, white wood nightstand, blue fabric curtain, white cotton pillow, white metal ceiling fan, and silver metal wall lamp. In front of me, I can see a white glass lamp, brown wood headboard, white cotton pillow, brown wood dresser, white wood bed, white cotton pillow, red metal wall lamp door, and brown wood drawer. Behind me, I can see a brown wood bureau, black glass television, brown wood stool, brown wood drawer, brown wood drawer, brown wood drawer.

Example:
User:
Description1: On the left, there is a white wood dresser, white glass lamp, and a white glass window/door. On the right, there is a white wood chair and a silver metal wall lamp. In front of me, there is an incomplete description. Behind me, there is a purple wood bed, yellow wood ceiling fan, white wood dresser, and another white glass window.
Description2: On the left, there is a white glass mirror, silver metal faucet, white porcelain sink, white wood bathroom sink, white wood bed, brown metal wall lamp. On the right, there are no specified items. In front of me, there is a silver metal faucet, white porcelain sink, white plastic toiletry, silver glass mirror, blue plastic soap toiletry, white bathroom sink countertop, blue metal wall lamp, black wood bathroom cabinet, black metal wall lamp. Behind me, there is a white porcelain tub, black cloth curtain, white cotton bed, brown wood bed, black cloth curtain, white porcelain bath.
These are depictions of what I observe from two different vantage points. Please assess the shared objects and spatial relationship in the descriptions to determine whether these two positions are indeed in the same place. Provide a response of True or False, along with supporting reasons.

Assistant:
Reasoning: To simplify the description, given the abundance of objects, we focus initially on common perceptions of larger objects, as they are less prone to detection errors.
Description1: On the left, there is a white wood dresser, and a white glass window/door. On the right, there is a white wood chair and a silver metal wall lamp. In front of me, there is nothing. Behind me, there is a purple wood bed, yellow wood ceiling fan, white wood dresser.
Description2: On the left, there is a silver metal faucet, white porcelain sink, white wood sink, white wood bed. On the right, there is nothing. In front of me, there is a silver metal faucet, white porcelain sink, white plastic toiletry. Behind me, there is a white porcelain tub, black cloth curtain, white cotton bed, white porcelain bath.
Shared Large Objects: Both descriptions feature common large objects, including a wood bed and a silver metal wall lamp, though the color of bed is different.
```

```
Spatial Relationships: Despite variations
in surrounding details, the consistent
mention of a white wood bed suggests a
shared spatial context, possibly from
different viewpoints within the same room.
 Minor differences, such as color
variations in the bed, may be attributed
to observational nuances rather than
indicating distinct rooms, reinforcing the
 likelihood of the same room.
Answer: True
```

```
Reasoning: To simplify the description,
given the abundance of objects, we focus
on common perceptions of object sizes.
Our attention is directed toward larger
objects, as these are less prone to
detection errors.
Description 1: On the left, there is brown
 wood headboard, white paper pillow. On
the right, black metal television, brown
wood dresser, and white wood closet
dresser are of relatively large size. In
front of me, there is a white wood bed
and a brown wood dresser. Behind me, I
can see a brown wood cabinet.
Description 2: On the left, there is white
 wood door. On the right, the white wood
bed, brown wood dresser, nightstand, and
ceiling fan are of relatively large size.
 In front of me, there is a white wood
bed, black glass television and a brown
wood dresser. Behind me, I can see a
brown wood drawer.
Shared Large Objects: the two descriptions
 exhibit significant commonalities,
prominently featuring large and easily
observable items such as a brown wood
dresser, a brown wood drawer, a white
wood bed, a metal wall lamp, a television
, and a brown wood headboard. Spatial
Relationship: the spatial relationships
within both descriptions remain
consistent, with the dresser positioned
near the bed in each scenario. Despite
minor variations in the color or material
 of smaller objects like stools or
curtains, these discrepancies appear more
 likely to stem from observational
nuances rather than indicating distinct
rooms.
Answer: True
```

E. OSG mapper: OSG updater module

Algorithm 1 UPDATELEAFNODES

Input: Object/Structure detections O_t^{det}, current Place node v^p, OSG $\mathcal{G}$
Output: Updated OSG $\mathcal{G}^{'}$, list of unassociated detections Ψ

1: // Get Object/Structure nodes in v^p
2: $\mathcal{V}_{nb} \leftarrow$ GETCONTAINEDNODES($v^p, \mathcal{G}$)
3: $\Psi \leftarrow$[]
4: **for** $v \in \mathcal{V}_{nb}$ **do**
5:   // Try to find matching detection for v
6:   `hasMatch`, $i_{match} \leftarrow$ DATAASSOCLLM(O_t^{det}, v)
7:   // Update attributes of v if matched with detection
8:   **if** `hasMatch` **then**
9:     $d^{label}, d^{desc}, d^{img}, \_ \leftarrow O_t^{det}[i_{match}]$
10:     $v_{label}, v_{desc}, v_{img} \leftarrow d^{label}, d^{desc}, d^{img}$
11:   **else**
12:     Ψ.append($O_t^{det}[i_{match}]$, $\mathcal{L}[i_{match}], i_{match}$)
13: **return** $\mathcal{G}^{'}$, Ψ

1) `ClassifyLayerLLM`: As object detector models detect both Objects and Structures without distinction, we query the LLM to sort the detections and assign them to the appropriate layer in the OSG (Algorithm 2 line 2). While we only require Objects and Structures at this step, we find that in practice the GroundingDINO object detector used in *Explorer* can occasionally produce detections of places like rooms (*e.g.* a bathroom). Thus our prompt asks the LLM to additionally filter out the Places from the detections, which are subsequently discarded.

```
 We observe the following: ["livingroom_0
", "window_13","door_2", "doorway_3", "
table_4","chair_5","livingroom sofa_6", "
floor_7", "wall_8", "doorway_9", "
stairs_10", "tv_16", "stool_17", "couch_18
", "remote_19"]. Please eliminate
redundant strings in the element from the
list and classify them into <PlaceType>, <
StructureType>, <Object> classes.
```

```
Answer:
room: livingroom_0
entrance: door_2, doorway_3, doorway_9
stair: stairs_10
object: table_4, chair_5, sofa_6,
window_13, tv_16, stool_17, couch_18.
remote_19
```

2) `DataAssocLLM`: To integrate Object and Structure detections into the OSG, we try to associate new detections with existing Objects and Structures in the OSG by querying the LLM (Algorithm 1 line 4). We prompt the LLM with the detected Object/Structure and its associated object features, and ask it to determine whether it matches an existing Object/Structure from the OSG based on similarity of object features.

Algorithm 2 OSG UPDATER

Input: Semantic information O_t, OSG specification $\mathcal{S}$, OSG $\mathcal{G}$, previous subgoal g_{t-1}, current estimated Place node v^p

Output: Updated state estimate s_t

1: // (1) Infer which layer each detection belongs to
2: $\mathcal{L} \leftarrow$ CLASSIFYLAYERLLM(O_t^{det})
3: // (2) Update, then add nodes in Layers 1-3 from O_t
4: **if** v^p is `None` **then**
5:     $v^p \leftarrow$ Add new place node to $\mathcal{G}$ with label O_t^{place}
6:     $\Psi \leftarrow$ `zip`(O_t^{det}, $\mathcal{L}$, `range`($|O_t^{det}|$))
7:     $\mathcal{I} \leftarrow \{\}$
8: **else**
9:     $v^p_{label} \leftarrow O_t^{place}$
10:     $\mathcal{V}_{upd}, \mathcal{I}, \Psi \leftarrow$ UPDATELEAFNODES(O_t^{det}, v^p, $\mathcal{G}$)
11: $\mathcal{V}_{new}, \mathcal{I} \leftarrow$ ADDLEAFNODES($\Psi, \mathcal{I}, \mathcal{G}$)
12: // (3) Add edges in Layers 1-3 from O_t
13: $\mathcal{V}^s_{new} \leftarrow$ GETSTRUCTURENODES($\mathcal{V}_{new}$)
14: ADDEDGES($v^p, \mathcal{V}_{new}$, "`contains`", $\mathcal{G}$)
15: ADDEDGES($v^p, \mathcal{V}^s_{new}$, "`connects to`", $\mathcal{G}$)
16: ADDEDGES($\mathcal{V}^s_{new}, v^p$, "`connects to`", $\mathcal{G}$)
17: **for** $v \in \mathcal{V}_{new} \cup \mathcal{V}_{upd}$ **do**
18:     $\mathcal{V}^{nb} \leftarrow$ GETNEIGHBOURNODES(O_t^{det}, $v, \mathcal{I}$)
19:     ADDEDGES($v, \mathcal{V}^{nb}$, "`is near`", $\mathcal{G}$)
20: // (4) Update, add nodes/edges in Layers 4-N
21: $v^{i-1} \leftarrow v^p$
22: **for** i in `range`$(4, N+1)$ **do**
23:     **if** LAYERCONNECTEDBYSTRUCTURE($i, \mathcal{S}$) **then**
24:         **if** ISSTRUCTURENODE(g_{t-1}) **then**
25:             $v^i \leftarrow$ ADDNODE($\mathcal{G}$)
26:             ADDEDGES(v^i, g_{t-1}, "`connects to`", $\mathcal{G}$)
27:             ADDEDGES(g_{t-1}, v^i, "`connects to`", $\mathcal{G}$)
28:         **else**
29:             $v^i \leftarrow$ GETPARENTNODE(v^{i-1})
30:     **else**
31:         $v^i \leftarrow$ INFERREGIONLLM($i, v^{i-1}, \mathcal{G}, \mathcal{S}$)
32:     ADDEDGES(v^i, v^{i-1}, "`contains`", $\mathcal{G}$)
33:     $v^{i-1} \leftarrow v^i$
34: **return** v^p

```
We are looking for <StructureType>/<Object
> that is near a tv, a chair and a stool.
Now we have seen the following <
StructureType>/<Object>: doorframe_2 that
is near chair and sofa. doorframe_3 that
is near a tv and a chair. wooden door_2
that is near table, sink and lamp.
Please select the <StructureType>/<Object>
 that is most likely to be the one I want
to find. If none of them seems likely,
reply None. Always follow this format:
Reasoning: <your reasoning>. Answer: <your
 answer>.
```

```
Reasoning: Among the given objects, "
doorframe_3" is mentioned to be near a TV
 and a chair, most likely meeting the
specified criteria of being near a TV,
chair, and stool.
Answer: doorframe_3
```

3) `InferRegionLLM`: We describe how we update region abstraction layers in the OSG (Algorithm 2 line 18). Some region abstractions are directly connected by Structures (*e.g.* floors are directly connected by stairs). In such cases, we are able to directly update the region abstraction layers by keeping track of the Structures encountered. However, some region abstraction layers may be abstract clusterings of lower layers - *e.g.* a supermarket may consist of "aisles", which are loosely grouped into "sections" that selling similar classes of items. "Sections" may not be explicitly connected by any Structure. To handle the general case where region abstractions are not connected by Structures, we query the LLM to determine whether the current location falls into an existing region abstraction node, or whether we a new node should be added.

```
Previously, we were in <AbstractionType>
containing the <PlaceType>, and we move
towards <Object/Structure subgoal>. Now we
 arrive in a <PlaceType>. Do you think the
 current state <PlaceType> belongs to [
list of existing <AbstractionType>s]?

If it belongs to any other existing region
 abstraction, return the region
abstraction name; otherwise propose the
name for this new <AbstractionType>
formatted as "<name> <AbstractionType> (
New)". Ensure that your response follows
the format:
Reasoning: <your reasoning>.
Answer: <your answer>
```

Concretely, we provide an example in a supermarket environment. The query below uses the supermarket OSG specification with an added region abstraction layer of "Section".

```
Previously, we were in the Cooked Food
Section containing the Deli Aisle, and we
move towards Freezer. Now we arrive in a
Dairy Aisle. Do you think the current
state Dairy Aisle belongs to any of the
existing region abstractions: ["Cooked
Food Section", "Clothing Section", "Dry
Goods Section"]?

If it belongs to any other existing region
 abstraction, return the region
abstraction name; otherwise propose the
name for this new section formatted as "<
name> Section (New)". Ensure that your
response follows the format:
```

```
Reasoning: <your reasoning>.
Answer: <your answer>
```

The response from GPT-3.5 is:

```
Reasoning: The Dairy Aisle typically
contains products such as milk, cheese,
yogurt, and butter, which are perishable
food items. Given its nature as a section
 primarily housing food products, it
doesn't fit within the existing region
abstractions provided ("Cooked Food
Section", "Clothing Section", "Dry Goods
Section").

Answer: Perishable Food Section (New)
```

4) Other functions: Aside from LLM-related functions, there are other key functions for updating leaf nodes. Once objects are assigned to appropriate layers and undergo data association function, they are divided into two groups. For objects already present in the OSG, the UPDATELEAFNODES function is utilized to update their stored information, including corresponding images, attributes, and connected edges. Conversely, for new objects, the ADDLEAFNODES function creates a new leaf node and adds its information and edges into the OSG.

Additional functions including GETSTRUCTURENODES, GETNEIGHBORNODES, GETCONTAINEDNODES, and GETPARENTNODES serve the purpose of retrieving specific nodes that meet the criteria given a node in the OSG. For instance, GETNEIGHBORNODES extracts all objects near the given node, i.e., connected by an "`is near`" edge to the given node.

F. Planner

1) Prompts for planning: The LLM reasons about the environment and proposes nodes to explore for the object goal based on the prompted OSG and OSG specification. We guide the LLM to plan hierarchically, following the hierarchical structure from the OSG specification. We do this as a simple means of encouraging hierarchical planning, and note that future extensions could employ the explicit hierarchical planning approach of [11].

```
You see the partial layout of the
environment: {"room": {"livingroom_1", "
connects to": ["door_1", "door_2"]}, "
diningroom_1": {"connects to": ["door_1
"]}}, "entrance": {"door_1": {"is near":
["towel_1"], "connects to": ["livingroom_1
", "diningroom_1"]}, "door_2": {"is near":
 [], "connects to": ["livingroom_1"]}}}
Question: Your goal is to find a <goal
object> sink. If any of the <LayerType> in
 the layout are likely to contain the
target object, specify the most probable <
LayerType> name. If all the <LayerType>
are not likely to contain the target
object, provide the <StructureType> (that
could connect <LayerType>) you would
select for exploring a new <LayerType>
where the target object might be found.


Example
User:
You see the partial layout of the
environment: {"room": {"livingroom_1": {"
connects to": ["doorway_1", "door_2"]}, "
entrance": {"doorway_1": {"is near": ["
table_1"]}, "door_2": {"is near": ["
clock_1"], "connects to": ["livingroom_1"
]}}} Question: Your goal is to find a oven
. If any of the rooms in the layout are
likely to contain the target object,
specify the most probable room name. If
all the rooms are not likely to contain
the target object, provide the door you
would select for exploring a new room
where the target object might be found.
Assistant:
Reasoning: There are only livingroom in
the layout. Among all the rooms,
livingroom is usually unlikely to contain
an oven, making it less likely for me to
find an oven in the current room. Instead,
 I plan to explore other rooms connected
to the current living room via entrances.
Evaluating the entrances, doorway1 stands
out as it is close to a table. Tables are
commonly found in kitchens, which often
contain ovens. Therefore, I have decided
to explore through doorway_1.
Answer: doorway_1
```

```
Reasoning: There is only a livingroom in
the layout. livingroom is not likely to
contain a sink, so I will not explore the
 current room. Among all the doors,
door_1 is near to the towel. A towel is
usually more likely to be near the
bathroom or kitchen, so it is likely that
 if you explore door_1 you will find a
bathroom or kitchen and thus find a sink.
Answer: door_1
```

```
In the bathroom, we observe ["mirror_2", "
lamp_1", "picture_7", "tool_6", "toilet_8
", "sofa_11", "floor_12", "wall_13"].
Please select one <Object> that is most
likely located near the <goal object> sink
```

```
. Always follow the format: Reasoning: <
your Reasoning>. Answer: <your answer>.
```

```
Reasoning: Among the given options, the
object most likely located near a sink is
 a "mirror." Mirrors are commonly found
near sinks in bathrooms for personal
grooming and hygiene activities.
Answer: mirror_2
```

2) Prompts for goal checking: While open-set object detectors and VQA models can identify a wide range of objects, they can provide unusual labels for uncommon objects. For instance, a beanbag may be identified as a “floor cushion”. To more robustly handle open-vocabulary queries with open-set VFMs, we propose to query LLMs to disambiguate whether a particular detection from the VFM is the goal object. This can result in a large number of LLM queries given the number of objects in real environments - as a compromise, *Explorer* queries the LLMs once at the start of a navigation episode to generate synonyms of the goal object, which are then matched against VFM detections.

TABLE VI
COMPARISON WITH VARIOUS OBJECT-GOAL NAVIGATION APPROACHES ON GIBSON VALIDATION SET. **TF** REFERS TO TRAINING-FREE APPROACHES. **NM** REFERS TO APPROACHES THAT BUILD NON-METRIC SCENE REPRESENTATIONS.

Method	Success (↑)	SPL (↑)	DTG (↓)	TF	NM
SemExp [2]	0.657	0.339	1.474	✗	✗
PONI [10]	**0.736**	**0.410**	**1.250**	✗	✗
FBE [17]	0.641	0.283	1.780	✓	✗
SemUtil [3]	0.693	**0.405**	**1.488**	✓	✗
Explorer-FMM	**0.734**	0.386	1.722	✓	✓

TABLE VII
ACCURACY OF USING OBJECT FEATURES TO MATCH AND DIFFERENTIATE OBJECTS, STRUCTURES AND PLACES

Node	Match	Differentiate	
		Same type	*Different type*
Objects	0.85	0.58	-
Structures	0.75	0.83	-
Places	0.88	0.80	0.94

G. Simulation experiment details

1) Fast Marching Method implementation: To isolate the effects of scene representation and reasoning, we use the same Fast Marching Method-based (FMM) controller across all LLM baselines and *Explorer* variants in simulation. Our implementation uses depth information and ground-truth pose from the simulator to build a 2D obstacle map, and is based on the implementation of [2]. It takes a goal specified in (x, y) coordinates, computes a shortest path with FMM over the metric map, then converts it to navigation actions. Our implementation additionally includes a heuristic recovery policy that attempts to rotate and perturb the robot in-place when it gets stuck or cannot find a valid path.

We adapt this FMM visuomotor policy to also take in *image goals*, for compatibility with *Explorer* variants. We assume that the simulated agent has access to depth images that are aligned with the RGB input being used by the OSG mapper. For each image goal, we can project the depth points that fall within its bounding box in the RGB-D image into 3D world coordinates, filter outlier 3D points and set the (x, y) coordinates of the centroid as the goal for FMM.

2) Object feature evaluation details: To determine if object features suffice for representing individual objects, structures, and places, we created a dataset containing object features. We randomly selected 10 environments in the HM3D validation dataset. In each environment, we randomly selected at least 6 place nodes, including the living room, dining room, kitchen, bedroom, and bathroom. If there are multiple rooms with the same room type, we include all of them. We have roughly 60 rooms and 150 objects in the dataset. For each place node, as well as all the structure and object nodes within the selected place nodes, we collected information about nearby objects from two different viewpoints, including their color, materials, and type. We assessed the feasibility of associating the same node from different viewpoints and distinguishing between different nodes in the dataset.

3) Scene graph construction evaluation details: To evaluate the quality of the OSGs constructed by the OSG mapper across various environments, we mapped 5 selected HM3D environments with the OSG mapper. We ran the *Explorer* system, but manually selected object subgoals for the agent to navigate to instead of using LLM planning, to enforce complete coverage of the environment. All environments are multi-storey, with four environments having at least two floors. Each of the environments has roughly 10 rooms.

H. Additional results for ObjectNav task

We evaluate Explorer-FMM in simulation on the full Gibson dataset. Results are shown in Table VI. On Gibson dataset, Explorer-FMM achieves a high success rate competitive with the strongest learned baselines like PONI, despite not having any ObjectNav or environment-specific training. Notably, we show stronger performance than other training-free approaches, like the state-of-the-art SemUtil that reasons semantically by combining classical planning with semantics.

I. Additional results for OSG

1) How effective are object features for data association?: We find that comparing *object features* with LLMs is an effective method of performing association and matching of objects, structures and places. We curate datasets of object features for objects, structures and places sampled across 10 HM3D scenes, with more details given in **??**. Table VII presents the accuracy achieved in matching and distinguishing objects, structures and places solely using object features. Across the diverse indoor household scenes tested, object

TABLE VIII
EVALUATION OF CONSTRUCTED SCENE GRAPH QUALITY

	Nodes		"contains"		"connects to"	
	Pr	**Re**	**Pr**	**Re**	**Pr**	**Re**
Floors	1.000	1.000	0.889	0.914	1.000	1.000
Rooms	0.846	0.880	-	-	0.771	0.831
Doors	0.845	0.800	-	-	0.783	0.857

features are informative and abundant enough to reach high levels of accuracy. We show strong performance on differentiating rooms of different types based on object features, to highlight that such features are a reliable alternative to localising with VQA models. Differentiating objects that are instances of the same type proves challenging since such objects are often clustered closely in self-contained groups in household environments and thus have similar object features, making it hard for LLMs to differentiate them. *E.g.*, chairs in a dining room are often grouped near a dining table, and sofas are often grouped around a coffee table. Even so, these objects' proximity means that wrong associations can still take us close to the correct object. Overall, the strong performance suggests that LLM-based object feature matching enables robust localisation and re-identification of observed objects and structures for navigation.

2) How well can we build OSGs over diverse environments?: We evaluate the accuracy of OSGs built with OSG mapper from teleoperated trajectories in HM3D homes by comparing them to human-annotated scene graphs. Table VIII presents precision and recall for mapping nodes for structures (doors), places (rooms) and abstractions (floors), as well as their outgoing edges. The strong performance indicates that most OSGs generated by OSG mapper accurately capture the scene's topology and contents.

Qualitatively, we show a constructed OSG for a two-storey house in Figure 4. The OSG recognises and correctly labels all key rooms in the house except for `livingroom_2`, which is a hallway that is misidentified by our VQA model. We demonstrate in Gibson's *Gratz* supermarket scene in Figure 5 to show how the OSG specification can help to capture the *right abstractions* for the scene, by specifying "aisles" as Place nodes.

3) How do OSGs enhance exploration with LLM planners?: We find that structured planning with LLMs, facilitated by rich, structured representations that encode the environment's layout, enhances exploration. From Table I, LFG improves significantly over Greedy LLM by providing richer and more comprehensive information (*i.e.* listing objects from accumulated metric map compared to listing objects within field-of-view) that is better structured (*i.e.* objects organised into spatial clusters compared to a flat list). OSG-based LLM planning improves further over LFG, which we attribute to OSGs being richer and more informative, and providing a scaffold for structured planning by organising scene elements with semantically meaningful abstractions. We show this with

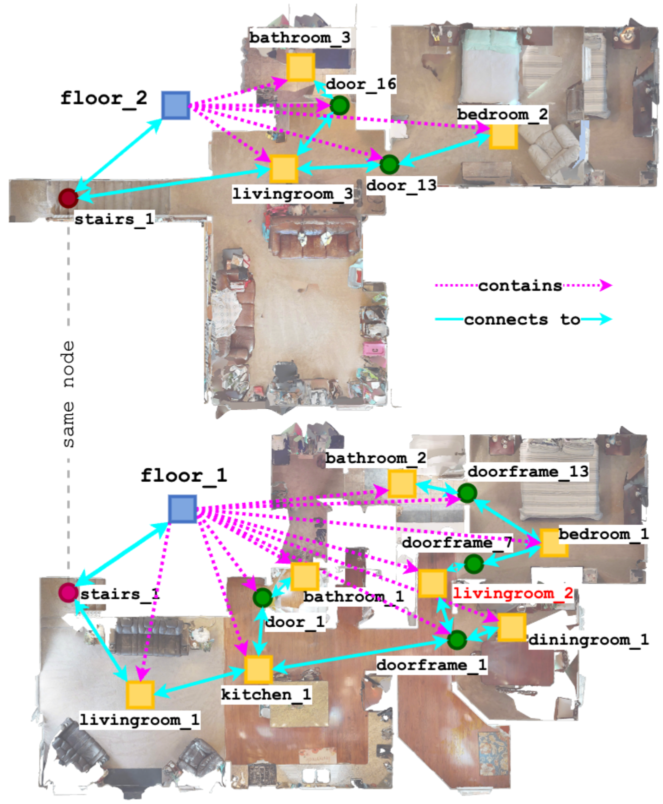

Fig. 4. OSG built with OSG mapper in two-storey house from HM3D validation set. `livingroom_2` (in red) is a hallway misclassified by the VQA model.

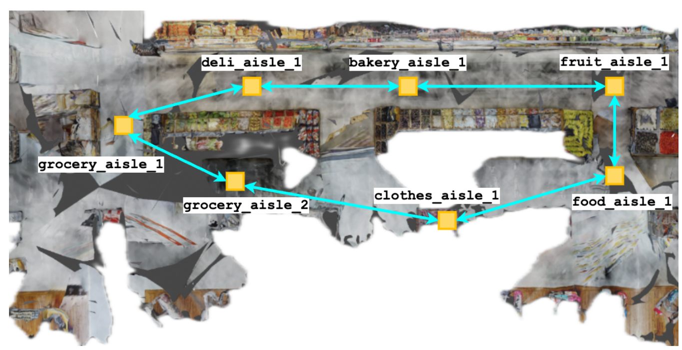

Fig. 5. Scene graph constructed in supermarket, using "aisle" Place nodes

qualitative examples of LFG and Explorer-FMM-GT's exploration trajectories in Figure 6.

In **(a)**, the robot starts in a bedroom and has to find a toilet. As there are few objects around, LFG defaults to geometric exploration and heads to a frontier on the left of the door, missing the bathroom immediately to the right of the bedroom. Since the OSG not only encodes objects but also place semantics, the LLM is able to reason that though the bedroom is unlikely to contain a toilet, a bathroom with one is likely to be nearby. The LLM decides to explore nearby, and the OSG specification-guided planning leads it to choose nearby "door" structures to explore, allowing it to quickly locate a bathroom and find a

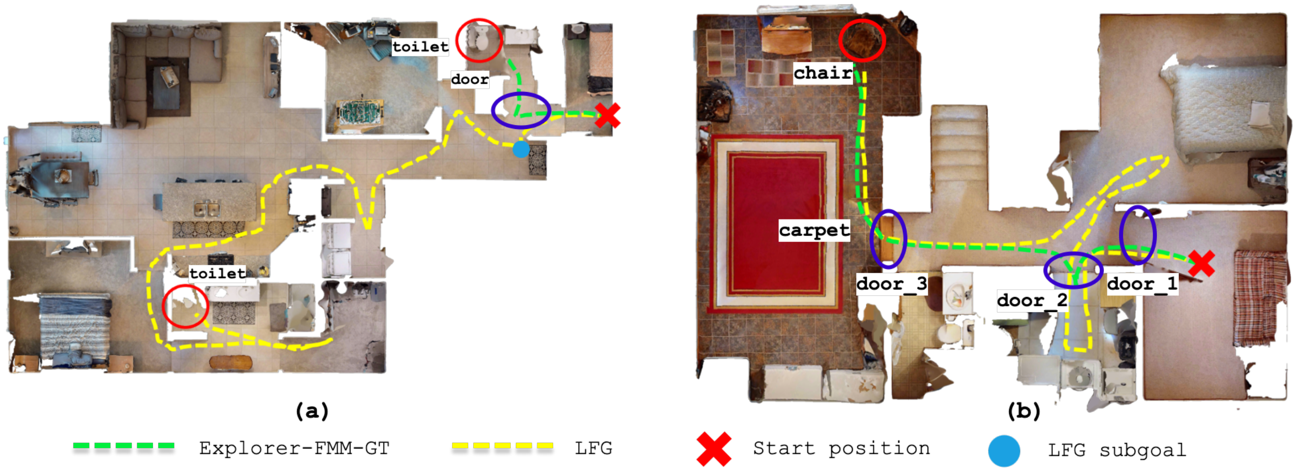

Fig. 6. Comparison of navigation performance between LFG and Explorer-FMM-GT

toilet. In **(b)**, the robot is tasked to find a chair. Both methods first enter the nearby "door_2" to explore the laundry room, which does not contain chairs. Due to the sparsity of objects, LFG is guided to exhaustively search over nearby areas. Since the OSG specification identifies "doors" as meaningful and important scene elements, this focuses Explorer-FMM-GT's attention on these, leading it plan to explore other "doors" recorded in its OSG memory after failing to find chairs in the laundry room. It previously saw "door_3" from the hallway and recorded the a carpet close to the door. It decides that this door is likely to lead to a living room-like space that may contain a chair and chooses to explore through there. Thus, the OSG specification helps to focus the planner's attention onto semantically meaningful goals for exploration, and the OSG serves as memory to enable efficient backtracking during search.

J. Additional results for OSG mapping

1) Examples of constructed OSGs: We provide additional examples of constructed OSGs from our tests described in Appendix G. We show OSGs for 3 different household environments in HM3D in Figure 7, Figure 8, Figure 9. From these tests, we observe that the constructed OSG generally achieves high accuracy. However, we also note that the OSG mapper has difficulty consistently identifying and relocalising to hallways, as it often attempts to localise using the objects in rooms observed from the hallway.

2) Importance of hierarchical structure: We highlight that LFG effectively serves as a single layer graph, as it contains only discrete object information, without capturing scene hierarchy or topology. In contrast, OSG additionally captures the place semantics and the environment's topology, enriching *Explorer*'s semantic reasoning. We note that this contributes to its ability to outperform LFG by reasoning hierarchically over places in addition to objects, as detailed in Appendix I3.

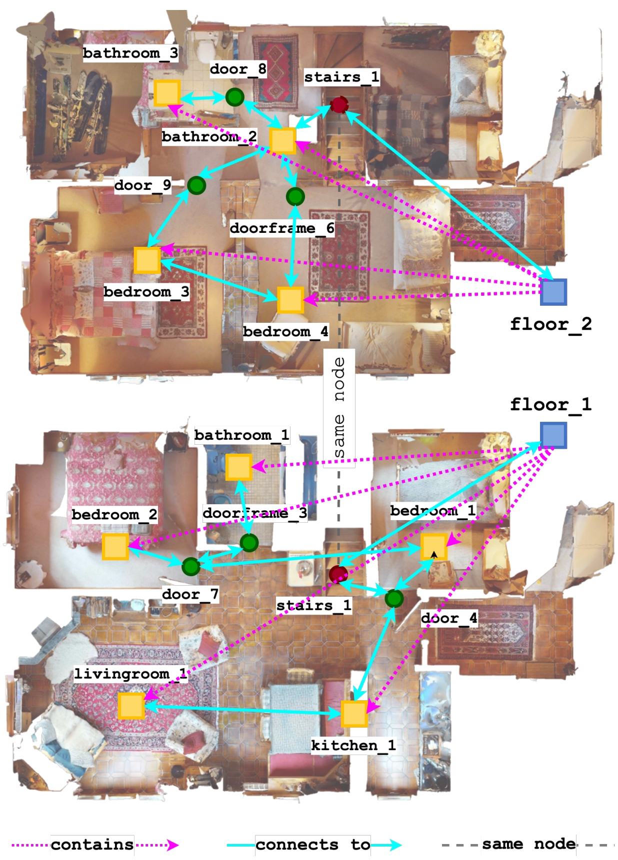

Fig. 7. HM3D Environment: 00835

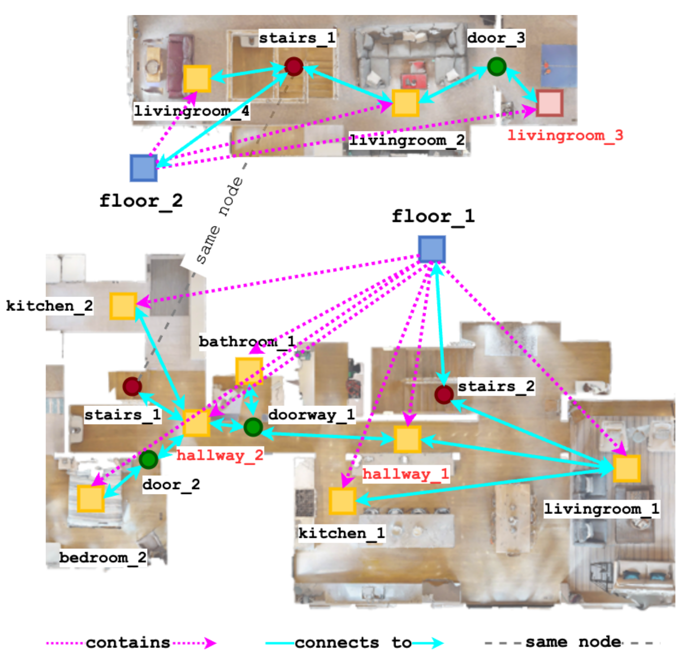

Fig. 8. HM3D Environment: 00814

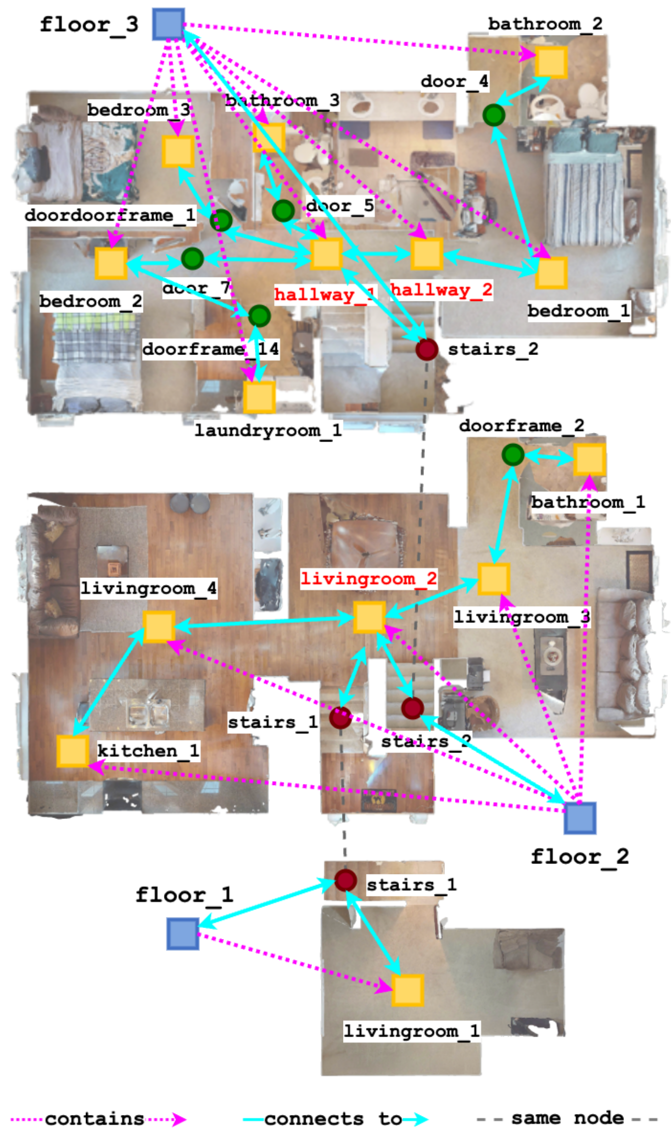

Fig. 9. HM3D Environment: 00832