GLEAM: Learning Generalizable Exploration Policy for Active Mapping in Complex 3D Indoor Scenes

Supplementary Material

A. Datasets

A.1. Data Creation & Data Filtering

There are several well-known datasets of indoor scenes, including ScanNet [11], Replica [44], Gibson [49], and Matterport3D (MP3D) [6]. However, the embodied AI community faces several challenges when working with these datasets: 1) there is a limited number of high-quality real-scan datasets, where "high-quality" refers to watertight surfaces, well-organized layouts, and unified reconstruction quality; 2) synthetic scenes often lack realistic features such as collision-rich layouts and high-fidelity furniture assets; 3) public datasets of indoor scenes employ different organizational structures, making it difficult to collect scene meshes with unified formats and scales, which impedes their effective use in simulation and policy learning.

In this work, we collect and preprocess 1152 high-quality meshes of complex indoor scenes from ProcTHOR-10K [12], HSSD [22], Gibson, and MP3D to learn a generalizable exploration policy for active 3D mapping. To leverage the digital assets in the general format originally coupling in AI2THOR [24] and Unity [20] backend, we created 972 meshes of complex scenes from ProcTHOR-10K using our export script. After manually filtering out the high-quality meshes of complex indoor scenes from these datasets, we preprocess them into a unified format and scale. Finally, these meshes are split into 1024 training scenes and 128 test scenes in our benchmark. The details of the data source are shown in Table 6. The details of preprocessing each dataset are as follows:

ProcTHOR-10K. ProcTHOR [12] and AI2-THOR [24] have empowered the research community to procedurally generate fully interactive, high-fidelity indoor scenes with diverse layouts for robotic training at scale. They introduce the ProcTHOR-10K dataset as templates of generated layouts, which includes 10,000 diverse indoor scenes. However, the original AI2-THOR platform and ProcTHOR dataset are limited by their reliance on the Habitat platform [41] and Unity Engine [20] for asset simulation and management, which constrains the extensibility of these valuable digital assets. To address this limitation, we developed an autonomous script that batch exports the generated scenes from Unity Editor to mesh files. This approach enables the procedural generation of scene meshes with editable content, including materials, floorplans, object placement, and controllable connectivity. Notably, users can generate an arbitrary number of scenes and then export them into mesh files using our exporting script. These infinitely generated 3D assets can be utilized for both policy training and digital content creation for AR/VR. We will release both the export script and the created assets.

Habitat Synthetic Scenes Dataset (HSSD). The HSSD dataset comprises 211 meticulously crafted 3D environments specifically designed to facilitate generalization ca-

Table 6. The sources of our collected and processed scene meshes. We created the meshes of complex scenes from ProcTHOR-10K using our export script. After manually filtering out the meshes of complex indoor scenes from **ProcTHOR-10K**, **HSSD**, **Gibson**, and **MP3D**, we preprocess them and split them into **1024 training scenes** and **128 test scenes**.

Dataset	Total Amount	Type / Mode	Amount	Training	Test
ProcTHOR-10K	896 (train) 76 (test)	4-room	284	256	28
		5-room	164	164	0
		2-bed-2-bath	280	256	24
		7-room-3-bed	114	96	18
		8-room-3-bed	28	28	0
		12-room	64	64	0
		12-room-3-bed	38	32	6
HSSD	32 (train), 10 (test)	easy	10	32	10
		medium	8		
		hard	24		
Gibson	96 (train), 24 (test)	real-scan	120	96	24
MP3D	18 (test)	real-scan	18	0	18
Total (Ours)	**1152**	**mixed**	**1152**	**1024**	**128**

pabilities within realistic 3D environments. This collection is characterized by its professionally curated digital assets and intricate spatial arrangements. We have selected 42 exemplary indoor scenes from this dataset, which serve as valuable photorealistic synthetic sources for exploration tasks. While decorative elements improve scene realism, they are unnecessary for policy learning and introduce excessive computational costs that hinder large-scale simulation training. Consequently, the scenes underwent systematic preprocessing through geometric simplification, particularly focusing on decorative elements and doors that might impede cross-room navigability.

Gibson & Matterport3D. Gibson and Matterport3D are public real-scan datasets providing hundreds of complex indoor scenes. However, the styles of these scene meshes exhibit significant variation, and the mesh quality is too inconsistent for direct use. Therefore, we filter these two datasets by the following criteria: 1) accurate reconstruction with minimal floaters and artifacts, 2) enclosed scene mesh with a nearly watertight external surface, and 3) one-floor structure. As a result, we obtain 120 diverse high-quality scene meshes from Gibson for training and evaluation. Also, we split all 18 selected meshes from MP3D for cross-dataset and out-of-domain evaluation.

A.2. Data Preprocessing

Mesh Preprocessing. To standardize the coordinate systems across scene meshes from different datasets, we transform all meshes such that their origin points lie at the geometric center of the floors, with the height direction isotropic to the +Z-axis. The transformation scripts are implemented in Python using Open3D library [55]

Ground-Truth Point Cloud. We generate ground-truth point clouds using the Poisson Disk sampling method [53], implemented in Open3D [55], to sample 100,000 points from the 3D scene meshes. To simplify visibility determination, we voxelize these point clouds at a specified resolution (grid size = 128 in this work) and filter out obviously invisible points, such as internal points enclosed within surfaces. These voxelized points serve as the ground-truth point clouds for the meshes and are used to compute key metrics like coverage ratio.

A.3. Dataset Split & Training Stages

Due to memory constraints and computational efficiency, we distributed the 1,024 training scenarios across two sequential training stages (i.e., stage 1 & stage 2). The final checkpoint from the first training stage served as parameter initialization for the subsequent stage. The one-stage exploration policy is optimized through 2.5k iterations and uses approximately 48 hours of training time on a single GeForce RTX 4090 GPU.

The dataset split of the two training stages is as follows. **Stage 1**: "procthor-4-room (256)", "procthor-5-room (164)", "procthor-8-room-3-bed (28)", "procthor-12-room-3-bed (32)", "hssd (32)". **Stage 2**: "procthor-2-bed-2-bath (256)", "procthor-7-room-3-bed (96)", "procthor-12-room (64)", "gibson (96)".

B. Implementation Details

B.1. Occupancy Grid Mapping Algorithm

The goal of an occupancy mapping algorithm [46] is to estimate the posterior probability of occupancy over voxels given the current probabilistic grid and the novel measurement event of camera ray casting. In particular, the more frequently a voxel is passed through by camera rays, the more confident the agent regards it as navigable free space.

PyCUDA-based Bresenham's Line Algorithm. Before updating the probabilistic occupancy grid, Bresenham's line algorithm [4] is implemented to cast the ray path in 3D space between the camera viewpoint and the endpoints among the point cloud back-projected from captured depth maps. To accelerate the computing efficiency, we use PyCUDA [23] to implement Bresenham's line algorithm.

Derivation of Map Updating. In practice, we adhere to the algorithm implementation outlined in GenNBV [9]. A comprehensive explanation of the methodology, along with the experimental results, is provided in the appendix of GenNBV. We provide the key derivation of the log-odds formulation of occupancy probability as follows:

Before updating the probabilistic occupancy map G_t, Bresenham's line algorithm is implemented to cast the ray path in 3D space between the camera viewpoint and the endpoints among the point cloud back-projected from D_{t+1}. According to the classical occupancy grid mapping algorithm [46], we have the log-odds formulation of occupancy probability:

$$\log Odd(m_i|z_j) = \log Odd(m_i) + \log \frac{p(z_j|m_i=1)}{p(z_j|m_i=0)}, \quad (4)$$

where m_i denotes the occupancy probability of i^{th} voxel in the map G_t, z_j is the measurement event that j^{th} camera ray passes through this voxel.

For the item $C = \log \frac{p(z_j|m_i=1)}{p(z_j|m_i=0)}$, there are only two cases for the measurement event in fact: $z_j = 0$ or $z_j = 1$. Thus, if the measurement event z_j (e.g., the voxel is passed through by the j^{th} camera ray) happens, we'll update the occupancy by adding the value of $C_1 = \log \frac{p(z_j=1|m_i=1)}{p(z_j=1|m_i=0)}$. If it's not passed, we'll add the value $C_2 = \log \frac{p(z_j=0|m_i=1)}{p(z_j=0|m_i=0)}$. The values of C_1 and C_2 can be set as empirical constants, depending on factors such as the accuracy of ray casting and the confidence of each ray. Actually, $C' = |\frac{C_1}{C_2}|$. We set a high value for C' (i.e., high confidence) because our experiments are based on the realistic simulator and accurate observations like depth maps.

Table 7. The key hyperparameters for our policy learning.

Term	Value
Optimizer	Adam
Optimization batch size	128
Learning rate	0.0001
Training Iterations	2500
Training Environments	32
N steps	512
N epochs	4
Buffer size	30
Value coefficient	0.8
Entropy coefficient	0.01
Discount factor γ	0.99
GAE τ	0.99
PPO clipping	0.2

Therefore, we can update the occupancy status of each voxel in the map G_t by adding a constant for each ray casting process. Note that the probabilistic occupancy map F^G is continuously updated within an episode. Finally, the occupancy status of voxels can be classified into three categories: unknown, occupied, and free, by setting an empirical threshold.

B.2. A* Path-Finding Algorithm

To evaluate the navigability between the agent's current position and predicted 3D target position, we implement a classic A* path-finding algorithm [17] in 3D space. We developed a CUDA-based implementation of the algorithm, increasing the computational efficiency. The system classifies a target position as unnavigable if the computed path length exceeds a predefined threshold, ensuring that our NBV policy predicts reliable and safe target poses.

Most previous works regard path-finding algorithms as a local policy and define a few movement commands (e.g., move forward $10cm$, turn left $30°$) as their action space. However, we don't follow this paradigm in our work for the following main reasons: 1) The key challenge of exploration policy is to determine the next best viewpoint, instead of the next neighbor step. Classic and learning-based planning and control methods both are capable of handling the control process toward the target viewpoint. 2) Popular action space, which consists of move forward $10cm$, turn left $30°$, turn right $30°$, makes redundant waypoints that produce inefficient trajectory, non-smooth control, and costly frequency of map updating, planning, and control.

B.3. Key Hyperparameters and Details

The key hyperparameters of our policy learning are shown in Table 7. Our implementation builds upon the codebase of Legged Gym [39] and utilizes the PPO implementation from Stable-Baselines3 [34], which is developed in PyTorch [33].

PPO Implementation. Specifically, given our parameterized policy π_θ, the objective of PPO is to maximize the following function:

$$L(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) \right], \tag{5}$$

where $A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)$ is the advantage function that measures the value of taking action a_t at state s_t under the current policy $\pi_{\theta_{\text{old}}}$. To prevent significant deviation of the new policy from the old policy, PPO incorporates a clipped surrogate objective function:

$$\begin{aligned} L^{\text{CLIP}}(\theta) =& \mathbb{E}_t[\min(\eta_t(\theta) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t), \\ & \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_{\text{old}}}}(s_t, a_t))], \end{aligned} \tag{6}$$

where $\eta_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and ϵ is a hyper-parameter that controls the size of the trust region.

Onboard Cameras. We assume that upon reaching the target pose, the agent performs four sequential 90-degree rotations and captures an observation at each orientation. To mitigate the significant computational overhead associated with repeated rendering during rotation, we implemented a simulation of four cameras mounted on the agent, with their headings oriented at 90-degree intervals.

Keyframe Budget During Inference. The budget $T = 50$ during inference was set based on the average exploration keyframes $\overline{T} = 31.78$ across methods. This value balances policy completeness and computational efficiency while not compromising the generalizability.

B.4. Implementation of Baseline Methods

We implement and evaluate the following works in our benchmark to demonstrate the superiority of our proposed method: 1) **Random Policy** randomly samples actions from Gaussian distribution within the action space. 2) **Vacuum** simulates a heuristic exploration policy for robot vacuums. We follow [8] to let policy move straight when safe and execute a random number of 9°turns when a collision occurs. 3) **FBE** always moves the agent towards the navigable nearest boundary between observed and unknown areas. 4) **UPEN [15]** estimates the information gain of candidate trajectories sampled by RRT [1], where the gain is estimated by model ensembles. We reduce the number of ensembling models and the number of layers due to the limited memory. 5) **ANM [51]** learns exploration in a neural implicit representation optimization framework. It estimates the information gain of candidate poses by three empirical criteria. We replace its RL-based local planner with our A-star planner. 6) **ANS [7]**: The original implementation of this policy relies on a global normalized map with unified resolution as input instead of an egocentric observed map, thus

it cannot be directly generalized to unknown environments. We adapt this policy to a generalizable pipeline that takes a global egocentric map as input and also augment the policy learning with our random initialization strategy to make it generalizable. Given ground-truth poses, we adapt the original active SLAM system to an active mapping system. 7) **OccAnt [35]**: Similar to ANS, we also provide ground-truth poses to adapt it to an active mapping system. Due to the limited storage, we reduced its map resolution and consequently increased the voxel size to ensure a similar perceptual range.

B.5. Visualization Implementation

All trajectories in Figure 1, 2, 6 are actual results. We record waypoints/keyframes and reconstruct the active mapping process using Open3D for offline visualization.

C. Training Strategy

C.1. Scene Updating Strategy

To enhance the generalizability, we create a training set including 512 diverse indoor scenes in each training stage from our GLEAM-Bench. However, we cannot launch such a large number of parallel training environments in simulation due to the limitations of computational efficiency and memory. As introduced in Sec. 4.4, we adopt a workaround to update the active scene in the limited training environments. We launch 32 training environments in Isaac Gym, and load 16 different scenes as a sampling set for each environment. During training, there is a predefined probability of p to randomly activate a scene in each environment's inactive sampling set. In particular, we move the replaced scenes to the inactive area (i.e., out of the agents' movement space) in the simulator and move the sampled scenes to the active areas of corresponding environments. As shown in Table 5, we found that frequently updating the active scenes utilizes the diversity of training scenes, and improves the generalizability of policies.

C.2. Capturing at Long-Term Target Positions

Previous work [8] typically employs discrete single-step actions, such as moving forward $10cm$ or turning left 10. However, this single-step planning and control approach is inconsistent with real-world robotic systems and significantly increases the computational cost of simulation for RL-based policy training. Moreover, real-world inference of this setup requires numerous policy network iterations, making it prohibitively time-consuming. Therefore, we optimize our approach to predict navigable next-best viewpoints in free space rather than relying on classical single-step actions.

To enhance practical effectiveness, we capture four surrounding views at each predicted position, simulating the scanning process. This multi-view setup provides a broader spatial context and enables more effective long-term planning during policy training.

D. Additional Results

D.1. The Reward of Trajectory Efficiency

Table 8. The effect of path efficiency reward. †: trained on 128 scenes and half-standard 2.5k iterations.

Settings	Cov.	AUC	Comp.	KF	TL
GLEAM †	60.23%	51.69%	0.89m	23.20	47.32m
GLEAM † with effi. rew.	56.61%	47.91%	0.96m	17.91	34.41m

As shown in Table 8, while implementing a generic efficiency reward term $r_t^{\text{Effi}} = -1$ [9] indeed reduce the number of keyframes (-5.29) and trajectory length (-12.91m), it penalizes exploratory actions like detouring around obstacles, leading to conservative policies (-3.62% Cov.).

D.2. Discussion and Future Directions

Realistic settings & Real-world deployment. 1) *Noisy observation.* Real-world deployments inevitably confront imperfect sensor inputs such as camera noise and depth ambiguity. Our framework employs probabilistic occupancy maps to mitigate noisy raw inputs of sensors, yet persistent noise patterns still propagate geometric errors during active mapping. 2) *Pose estimation.* While our system obtains accurate poses through the simulator, the pose estimation in practical scenarios with fast camera motions or textureless regions induces pose drift. This spatial uncertainty manifests as misaligned geometry fragments, particularly when scanning thin structures like chair legs or lamp arms. 3) *Open environments.* Unlike bounded scanning domains in simulation, real-world scenes often contain dynamically expanding areas (e.g., newly opened doors). Existing frameworks struggle to build memory-efficient representations for unbounded and dynamic scenes, which shows that the sim-to-real gap still has a lot of potential to be explored.

While real-world deployment remains an open challenge, we advance the sim-to-real validation across *sensor-*

Table 9. The robustness of GLEAM under Gaussian noise $N(0, \sigma^2)$ (unit: meter) during inference. $\boldsymbol{\sigma_P^2}$: variance of cumulative pose noise. $\boldsymbol{\sigma_D^2}$: variance of depth noise. **KF**: keyframes. **TL**: trajectory length. **"*"**: non-cumulative per-step noise

Settings	$\boldsymbol{\sigma_P^2}$	$\boldsymbol{\sigma_D^2}$	Cov.	AUC	CD	KF	TL
GLEAM (no noise)			66.50%	57.63%	0.80m	29.57	54.51m
Pose-only	0.1	0	63.27%	54.41%	0.86m	28.27	41.63m
	0.3	0	60.73%	52.11%	0.92m	24.95	35.71m
	0.5	0	55.59%	48.24%	0.99m	20.31	22.79m
	0.1*	0	66.18%	56.62%	0.76m	30.69	49.36m
	0.5*	0	58.54%	50.53%	0.95m	23.61	38.51m
Depth-only	0	0.05	64.94%	55.31%	0.82m	30.44	45.05m
	0	0.1	60.21%	51.09%	0.96m	29.21	36.78m
	0	0.2	54.77%	46.30%	1.13m	27.51	27.21m
Depth+Pose	0.1	0.05	60.44%	51.76%	0.91m	28.20	36.62m
	0.3	0.1	54.03%	47.21%	1.09m	24.74	24.33m
	0.5	0.1	51.36%	44.32%	1.17m	20.94	20.56m

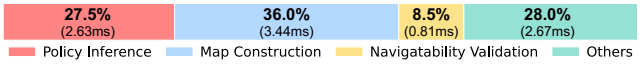

Figure 7. The one-step inference time of our key components.

noise tolerance and *computation cost* to demonstrate concrete progress toward deployability. To evaluate the sensor-noise tolerance of GLEAM, we inject hardware-aligned Gaussian noise to observations during inference, deriving from real-world sensors like Intel®RealSense D455 depth camera ($< 2\%$ error at $4m$) and TDK InvenSense ICM-42688-P IMU. As shown in Table 9, GLEAM maintains strong robustness despite training on ideal observations, which stems from our probabilistic map that inherently suppresses transient noise by Bayesian updating.

Our system achieves real-time inference (**104.7Hz**) on a PC with an RTX 3090 GPU, with latency analysis in Figure 7 demonstrating the efficiency of our lightweight policy network and CUDA-accelerated map updating/A* planning, ensuring seamless high-frequency perception and decision-making in the real world.

Challenging 3D benchmark & 3D action space. We've explored the potential of our framework for challenging 3D benchmark, including 3D action space ($x, y, z, pitch, yaw$) and 3D optimization objectives. In this setting, our agent is encouraged to capture all details, such as the surface under the underside of a table, in complex 3D scenes. While our agent demonstrates promising effectiveness and generalizability in scanning most of unobstructed surfaces, we found it quite difficult to capture the geometrically complex surfaces (e.g., the undersides of tables and self-occlusion surfaces of decorations) in cluttered 3D environments. Also, the heavy computational burden and limited memory prevent us from optimizing the components like 3D representations and scaling the number of training scenes.

Challenging multi-floor complex scenes. Although we've been exploring active mapping for complex single-floor indoor scenes, the tough cases in the real world are the multi-floor indoor scenes. These environments introduce unique cross-floor topological dependencies and vertical navigation constraints that existing frameworks fail to adequately model. Moreover, the inherent geometric discontinuities between floors exacerbate memory fragmentation when using conventional spatial representations, leading to increasing memory overhead.

Multi-agent collaboration. Multi-agent collaboration is one of the solutions for complex scenarios such as multi-floor scenes. However, scaling to collaborative active mapping introduces novel fundamental challenges in distributed strategy optimization, dynamic role allocation, and communications. These challenges demand the rethinking of existing frameworks, particularly in developing scalable and memory-efficient representations and decentralized decision-making architectures.