

Goal-driven 3D Map Generation with Reinforcement Learning

Lorenzo Baggi Marco De Negri Vassili De Palma Alessio Salvatore

Supervisors: Daniel Barath, Jelena Trisovic

Abstract

3D maps are fundamental for many robotic tasks, but storing them at a uniform fine resolution is wasteful: most of the scene never matters for the task at hand. Existing adaptive-mapping methods either rely on hand-tuned heuristics or ignore what the map is actually for. We instead frame resolution allocation as a sequential decision problem and train a reinforcement-learning policy that dynamically refines or coarsens the map under an explicit memory budget, optimising directly for downstream navigation rather than reconstruction fidelity. Built on top of PEANUT and evaluated in Habitat, our learned policy retains 79% point-goal success rate while using 87% less memory on average than a uniformly fine map, and matches the usage of an all-coarse map at far higher success.

1. Introduction

Autonomous systems rely on 3D maps to perceive, navigate, and interact with their environment. Volumetric representations based on Truncated Signed Distance Fields (TSDFs) [2–4] have become the standard for dense reconstruction. However, conventional TSDF pipelines employ a fixed voxel resolution, which leads to high computation and storage demands. Recent work has therefore shifted toward adaptive mapping, where voxel resolution is adjusted according to scene complexity. MAP-ADAPT [12] represents an important step in this direction, combining geometric cues and manually defined semantic classes to control voxel subdivision. By allocating higher resolution to semantically meaningful regions, it produces multi-resolution 3D voxel maps. However, this design rests on a static, task-agnostic assumption: the importance of a region is fixed in advance by a hand-tuned class-to-resolution list and a geometric-complexity threshold. Such rules require expert tuning, carry no notion of what the map is for, and do not allow one to easily define a budget for the 3D reconstruction of the environment. We address these limitations by formulating adaptive map resolution as a sequential decision process. At each timestep, a reinforcement learning agent observes the current state of the map, the task speci-

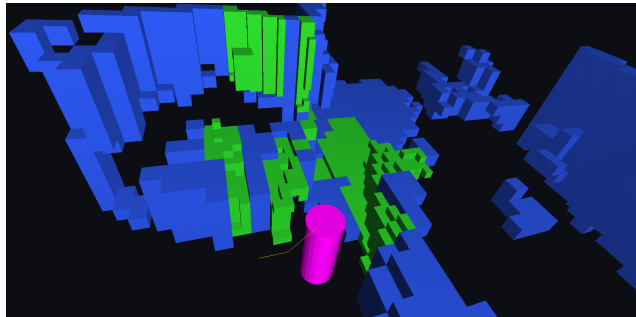


Figure 1. Open3D visualization on HM3D episode 3 point-goal. Blue regions indicate coarse resolution, green regions indicate medium resolution, and red regions indicate fine resolution (none needed in this task).

fication and decides how to modify the 3D representation, refining regions where added detail advances the task and coarsening regions where it does not, subject to an explicit user-defined memory budget. Rather than optimizing reconstruction fidelity in isolation, our agent is trained against the downstream task: it allocates resolution so as to maximize navigation performance under a fixed storage constraint. This yields adaptive maps that respond dynamically to changes in the scene and across different tasks, replacing hand-engineered class heuristics with a single learned, budget-aware policy. We build on PEANUT [11], a navigation agent that maintains a dense, uniformly fine-resolution voxel map, to complete PointGoal and ObjectGoal navigation tasks in the Habitat simulator [8]. We replace its dense grid with a sparse, hierarchical, multi-resolution map and train a policy that allocates resolution for the navigation task rather than for reconstruction fidelity, and we show that this task-driven policy (see Figure 1) nearly matches the navigation success of a uniformly fine map while using only a fraction of its memory.

2. Related Work

Adaptive Mapping. To improve efficiency, adaptive mapping techniques adjust voxel resolution to scene content. Geometry-based methods refine regions of high curvature or structural complexity and coarsen flat areas, im-

proving geometric fidelity near surfaces but ignoring semantic relevance. Semantic-based methods add object information to guide subdivision: Panoptic Multi-TSDFs [9] reconstruct separate TSDF volumes per instance, while MAP-ADAPT [12] combines geometric and semantic cues into a single multi-resolution TSDF map by refining user-specified semantic classes. MAP-ADAPT demonstrates real-time adaptive reconstruction and forms the basis of our map representation, but its allocation rests on manually defined class-to-resolution rules that are fixed offline, do not generalize to unseen categories, and crucially carry no notion of which regions matter for a given task.

Learning-Based Adaptivity. Reinforcement learning has been applied extensively to active exploration and SLAM, where a policy selects actions or viewpoints that improve coverage or localization [1, 6]. These methods learn how to move to acquire information, but leave the underlying map a passive, fixed-resolution structure. UNRL [5] is, to our knowledge, the first to apply reinforcement learning to the structure of the map itself, replacing MAP-ADAPT’s class lists with an entropy signal over geometric, semantic, and texture cues and learning a budget-aware subdivision policy; its reward, however, targets reconstruction fidelity, so resolution is spent wherever the scene is hardest to reconstruct. We adopt the same budget-aware RL view of map adaptation, but optimize the policy for a downstream navigation objective. We instantiate it within PEANUT [11], an ObjectGoal navigation method that predicts unseen target locations from a top-down semantic map and plans toward them while relying on a dense, uniformly fine-resolution voxel map. Using PEANUT as our backbone, we show that an RL policy can reallocate this resolution toward task-critical regions, retaining navigation success at a fraction of the memory.

3. Method and Contributions

3.1. Pipeline Overview

We build our agent on top of PEANUT [11], and our core contribution is to replace its dense 3D map with a *sparse, hierarchical, adaptive-resolution* voxel map whose resolution is controlled by a learned policy under an explicit memory budget. The complete perception–mapping–planning pipeline runs as a closed per-frame loop, illustrated in Fig. 2: the simulator emits a new RGB-D frame, the 3D map is updated, a reinforcement-learning policy adapts the map resolution, the 3D map is projected down to a 2D top-down map, the 2D map drives planning, and the resulting action moves the robot, producing the next frame.

The pipeline supports two navigation settings. In *Point-Goal* navigation the agent must reach a fixed target coordinate, while in *Object-Goal* navigation (ObjectNav) it

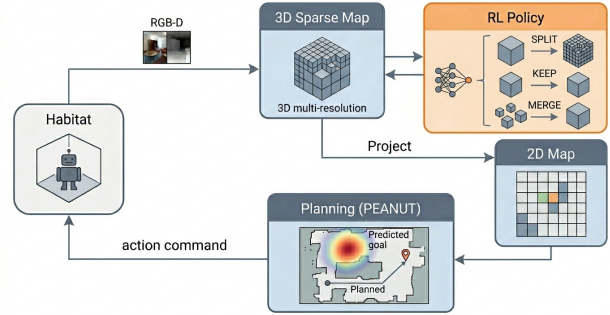


Figure 2. End-to-end perception–mapping–planning pipeline. The simulator (Habitat), semantic perception, and prediction–planning stages are reused; the sparse hierarchical 3D map, the adaptive-resolution policy, and the incremental 3D→2D projection are our contribution.

must reach an instance of a target object *category* (e.g. *bed*, *toilet*, *sofa*) in a previously unseen scene. The loop is identical in both cases; the only difference is that in the Point-Goal setting the goal is a known location, so the semantic-segmentation and target-prediction stages are skipped, while the 3D map, the resolution policy, and the 2D projection are unchanged.

3.2. Our Contributions

Our contribution spans three stages of the loop: the 3D map, the resolution policy that controls it, and the incremental projection that keeps the 2D map in sync.

Sparse hierarchical 3D map. PEANUT represents the scene with a *dense* voxel grid produced by trilinear splatting, whose memory cost grows with the whole bounded volume regardless of how much of it is actually occupied. We re-implement the map from scratch in Python as a sparse, hierarchical, multi-resolution grid (GlobalSparseGrid). Only observed voxels are stored, so memory scales with the observed surface rather than the volume; every voxel lives at one of several resolution levels (5 cm / 10 cm / 20 cm), and the insertion logic routes each point so that at most one voxel covers any given position across all levels; and the map is accumulated persistently in world coordinates over the whole episode rather than rebuilt per frame. For each frame the depth image is back-projected into a point cloud, transformed into a global world frame (in centimetres) using the GPS+compass pose, and inserted into the grid.

Per-voxel evidence, carving, and pruning. Rather than a single occupancy value, every voxel accumulates the evidence observed for it over the whole episode: a *hit count* (how many depth points fell inside the voxel, acting as an occupancy confidence), the per-class semantic scores summed over those points (whose mean recovers a class

probability), a *free-space count* (how many camera rays passed through the voxel without stopping in it), and an RGB colour sum for visualisation. Keeping both a hit count and a free count lets us reason about occupancy as a ratio of evidence for versus against, instead of a hard binary flag. On this basis we implement free-space carving: for each depth hit we sample points along the ray from the camera to the hit and increment the free-space count of every voxel the ray traverses before stopping. A voxel repeatedly seen empty, specifically one whose free count exceeds its hit count, is pruned, since it is most likely a boundary or noise artifact rather than a real surface. Pruning keeps the sparse map clean and its memory footprint low over a long episode.

Adaptive-resolution policy. Between the map update and the 2D projection, a learned reinforcement-learning policy adapts the resolution of the voxels, refining selected regions (SPLIT), leaving others unchanged (KEEP), or collapsing fine voxels into coarser ones (MERGE), so as to keep the most useful detail while respecting a fixed memory budget. The policy writes these changes directly back into the 3D map, which is what triggers the re-projection described next. We defer the full formulation, i.e. the observation, the score-driven action, the reward, and training, to Sec. 4.

Incremental 3D→2D projection. Planning operates on a 2D top-down map (obstacle, explored, and per-class semantic channels), so the updated 3D map must be projected to 2D. A naive implementation would re-project the entire grid on every frame; instead we develop an incremental “delta-buffer” projection. The buffer maintains a reverse index from each 2D pixel to the set of voxels currently covering it; when a voxel is inserted, erased, split, or merged, its footprint pixels are added to a *dirty set*. At projection time we iterate only over the dirty pixels, recompute each as the maximum over the live voxels still in its column, and clear the set. The projection further applies Z-band gating: the obstacle and semantic channels receive contributions only from voxels within a waist-height band, while the explored channel (and a few tall classes) accumulates over the full column. This keeps the 2D map an exact, always-consistent projection of the live 3D voxels while only paying for what actually changed.

3.3. Reused Components

The remaining stages are reused essentially as provided, and frame our contribution rather than being part of it.

Simulation environment. The environment is Habitat [8] (Habitat-Lab + Habitat-Sim), with the task being ObjectGoal Navigation on the HM3D [7] scene dataset using the Habitat ObjectNav Challenge episodes. At every step Habitat provides an aligned RGB-D frame, the agent pose via GPS and compass sensors, and the target object category; it also executes the discrete navigation actions (move

forward, turn left, turn right, stop) and reports task metrics such as success and distance to goal.

Semantic perception. In the Object-Goal setting each RGB frame is passed through a Mask R-CNN semantic-segmentation network that produces per-pixel class probabilities, which together with the depth image are lifted into the map; the model and its inference code come from the PEANUT stack. In the Point-Goal setting this stage is skipped and the per-class channels are left empty. The geometric projection utilities (depth-to-point-cloud back-projection and pose handling) are likewise reused from PEANUT.

Prediction and planning. The 2D map is consumed by PEANUT’s prediction-and-planning module: a prediction model estimates where the unseen target object is likely to be given the partial 2D semantic map, and a Fast Marching Method (FMM) planner computes a geodesic distance field over the traversable space, selects a long-term goal, and produces the next navigation action. The selected action is sent back to Habitat, which moves the robot and emits the next frame, closing the loop; the episode ends when the agent issues *stop* near the target (success) or the step budget is exhausted. Finally, the RL training machinery, i.e. the PPO [10] implementation from stable-baselines3 and the Hydra configuration, is reused as well; only the MDP, score, action, and reward of Sec. 4 are ours.

4. Policy Formalization

We cast online resolution selection as a sequential decision problem in which an agent interacts with the adaptive map. At each step the agent observes a compact global summary of the map and decides, through a pair of score thresholds, which regions to refine and which to compress, refining the parts that matter for the task and coarsening everything else. The intended behaviour is a trade-off: the agent should preserve the route to the goal at high resolution while keeping the overall map small enough to respect the memory budget. We formulate the task as a Markov decision process (MDP) and train the policy with Proximal Policy Optimization (PPO) [10].

4.1. Problem Formulation

We model resolution selection as an MDP defined by the tuple $M = (S, A, p, r, \gamma)$. The *environment* comprises the underlying navigation agent (PEANUT [11]) and the hierarchical adaptive map, whose voxels are stored at one of three levels: fine (L_0 , 5 cm), medium (L_1 , 10 cm), and coarse (L_2 , 20 cm). The basic decision unit is a *super-block*, a $4 \times 4 \times 4$ cube of L_2 voxels (80 cm side). A state $s \in S$ encapsulates the full state of the adaptive map. An action $a \in A$ is a pair of ordered thresholds on the per-block score (Sec. 4.2); these thresholds deterministically induce, for every occupied super-block, one of three operations, SPLIT,

KEEP, or MERGE, which respectively refines, preserves, or compresses the block. The transition p applies these operations and advances the navigation agent, which observes the next frame and updates the map accordingly. Each transition yields a reward $r(s, a)$ that rewards staying just below the memory budget while penalising any damage to the planned route, with future rewards discounted by γ . Feeding the policy the complete state, i.e. every stored voxel and its score, is not feasible: the number of occupied blocks varies over time and across scenes and can be very large, so a per-voxel input would have a variable, potentially intractable dimension. Instead, the agent receives a fixed-size observation $o \in O$ that approximates the map state, and operates as $\pi_\theta : O \times A \rightarrow \mathbb{R}$. The observation is a global vector of 21 values:

$$o_t = [f_{\text{clip}}, b, \underbrace{h_1, \dots, h_{16}}_{\text{score histogram}}, \phi_{\text{fine}}, \phi_{\text{med}}, \phi_{\text{coarse}}]. \quad (1)$$

Here $f_{\text{clip}} = \min(2, \text{used}/\text{budget})$ is the memory fill, the main dynamic signal, which equals 1 exactly at the budget; b is the absolute budget, log-normalised to $[0, 1]$ between a minimum of 1 kB and a maximum of 100 MB; it is constant within an episode but may vary across episodes, and is included so that a single policy can adapt its behaviour to different budgets; $h_1, \dots, h_{16}$ is a 16-bin histogram of the per-block scores (Sec. 4.2), normalised to sum to one, describing how task-relevance is distributed across the map; and $\phi_{\text{fine}}, \phi_{\text{med}}, \phi_{\text{coarse}}$ is the current fraction of voxels stored at each resolution level over the whole map. By design, the observation is independent of the number of blocks, keeping the policy network small and cheap to evaluate.

4.2. Score and Action

The policy never decides on individual blocks directly. Instead, it reasons over a single scalar *score* that ranks every block by how much it is worth refining, and its action picks two thresholds that split this score range into three regions, each mapped to a fixed resolution change. We describe the two ingredients in turn.

Score. We assign every occupied super-block a score in $[0, 1]$ that captures how task-relevant the block is,

$$\text{score} = \text{clip}(0.8 \cdot \text{on_path} + 0.2 \cdot \text{coarseness}, 0, 1). \quad (2)$$

The term $\text{on_path} \in \{0, 1\}$ flags blocks lying on the route corridor from the robot to the goal, computed with a Fast Marching Method (FMM) planner on the geodesic distance and dilated by the robot radius; it is the dominant term (weight 0.8). The term $\text{coarseness} \in [0, 1]$ measures the refinement headroom of *that specific block*, i.e. how much it could still be refined, computed from the block’s own resolution composition as $\text{coarseness} = 0.5 \phi_{\text{med}}^{\text{blk}} + 1.0 \phi_{\text{coarse}}^{\text{blk}}$, where $\phi_{\text{med}}^{\text{blk}}$ and $\phi_{\text{coarse}}^{\text{blk}}$ are the fractions of medium and

coarse voxels *inside the block*; a coarse block has more headroom than a medium one, hence the larger weight. This term carries little weight (0.2) and only serves to break ties among off-route blocks. As a result, on-route blocks score high (≥ 0.8) and off-route blocks score low (≤ 0.2), so the score arranges blocks along a single axis from “keep at all costs” to “safe to compress”.

Action. The action of the policy is to place two thresholds on this score axis, $0 \leq c_1 \leq c_2 \leq 1$, which partition the score range into three regions. Concretely, the network outputs three logits, mapped through a softmax to weights $(w_{\text{merge}}, w_{\text{keep}}, w_{\text{split}})$ that sum to one, from which the thresholds are read off as $c_1 = w_{\text{merge}}$ and $c_2 = w_{\text{merge}} + w_{\text{keep}}$. The same two thresholds are then applied to all occupied super-blocks at once: every block whose score falls in a given region undergoes the same resolution change, namely MERGE (compress), KEEP (leave unchanged), or SPLIT (refine),

$$\text{block} \mapsto \begin{cases} \text{MERGE} & \text{if score} < c_1, \\ \text{KEEP} & \text{if } c_1 \leq \text{score} < c_2, \\ \text{SPLIT} & \text{if score} \geq c_2. \end{cases} \quad (3)$$

Concretely, a SPLIT promotes every voxel in the block by one resolution level (e.g. a medium voxel becomes eight fine voxels), while a MERGE demotes it by one level; voxels already at the finest level L_0 are unaffected by a SPLIT, those at the coarsest level L_2 by a MERGE, and a KEEP leaves the block untouched. The occupancy and feature information stored in the block is correspondingly redistributed across the newly created finer voxels on a SPLIT, or aggregated into the coarser voxels on a MERGE. The action thus has only three dimensions regardless of map size, keeping the policy compact while still controlling the global allocation of resolution. Because route membership enters only through the score and never acts as a hard constraint, a single well-placed cut is enough to keep the route at fine resolution while compressing the rest, but it is the policy that must *learn* where to place that cut.

4.3. Reward

Because our framework is based on RL, the reward can integrate non-differentiable feedback about memory and route quality that would be hard to optimise directly. The per-step reward sums three terms, a budget reward and two route-damage penalties:

$$r = \underbrace{\Lambda(f)}_{\text{budget}} - \underbrace{0.5 \text{clip}(\Delta \ell, 0, 50)}_{\text{route lengthening}} - \underbrace{(0.5 u_t + 2.0 (u_t - u_{t-1}))}_{\text{route blocking}}. \quad (4)$$

Budget. Here $f = \text{used}/\text{budget}$ and the tent-shaped function Λ rises linearly to its maximum of +1 just below

the budget and drops steeply once the budget is exceeded,

$$\Lambda(f) = \begin{cases} \frac{1}{0.9} f & f \leq 0.9, \\ 1 - 10(f - 0.9) & f > 0.9. \end{cases} \quad (5)$$

This encourages the policy to fill memory until it nearly saturates the budget, i.e. to refine where it helps, while making any overshoot very costly, e.g. $\Lambda(1.1) = -1$ and $\Lambda(1.5) = -5$.

Route lengthening. $\Delta\ell$ is the increase in the planned path length to the goal caused by the current step, obtained by comparing the FMM path before and after the SPLIT/MERGE operations. Small detours incur a small penalty, while actions that fully block the route yield $\Delta\ell = \infty$, capped at 50 for a maximum penalty of $0.5 \times 50 = 25$. This discourages compressing blocks that would thicken an obstacle across a doorway and lengthen the path.

Route blocking. $u_t \in \{0, 1\}$ indicates whether the goal is unreachable after the current step. The fixed cost $0.5 u_t$ discourages remaining in a blocked state, while the potential-based term $2.0(u_t - u_{t-1})$ provides a gradient that actively rewards reopening the route ($1 \rightarrow 0$ gives $+2.0$) and penalises breaking it ($0 \rightarrow 1$ gives -2.0). The potential term is essential: a constant penalty alone is uninformative once the route is already broken, whereas the difference term pushes the policy to restore reachability and is the strongest single signal in the reward.

5. Results and Discussion

5.1. Training Setup

We train the policy with PPO [10] using a discounting factor 0.99, learning rate 2×10^{-4} , rollouts of 250 steps, batch size 64, 2 epochs per update, clip range 0.2, entropy coefficient 0.01, and target KL 0.02. Training uses a single fixed episode: a point-goal navigation task in which the goal is the position of the bathroom, with success declared within 0.1 m of the goal, and the same episode is replayed at every reset. The setup is designed so that resolution allocation is the decisive factor in solving the task. New voxels are initialised at the coarse level (L_2), so an unrefined corridor fails because doorways effectively close at low resolution; this creates the pressure that makes the route terms pay the policy for opening the corridor, while the budget tent forces it to keep everything else coarse. The budget of 8 MB is binding but not degenerate: it is tight enough that the map cannot simply store everything at fine resolution and some compression is required, yet loose enough that a non-trivial learned allocation exists.

5.2. Training Results

After 200k env steps (3181 episodes and approximately 67 hours on a single environment on a NVIDIA RTX 2080Ti)

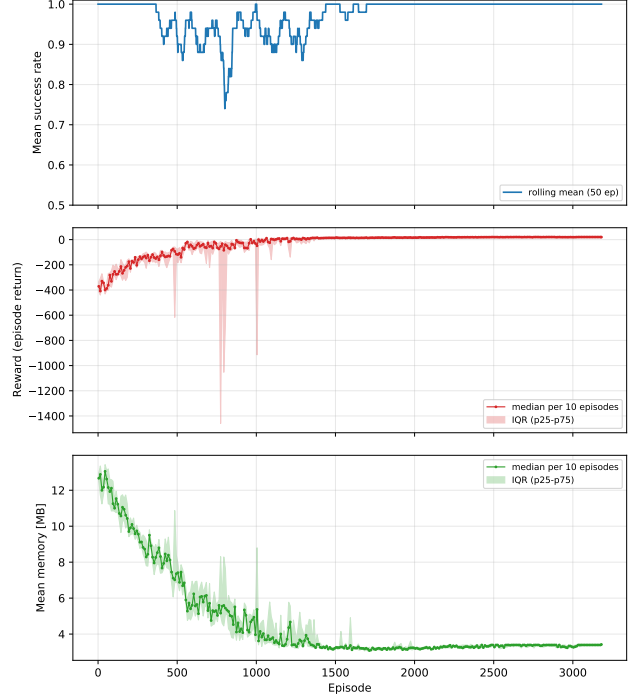


Figure 3. Training curves. Top: mean success rate. Middle: median episode reward. Bottom: median memory usage.

the policy stabilized. In the first phase of training the policy is still close to random and allocates too much resolution for the given budget: this yields a policy that always succeeds but at an inappropriate memory cost, and therefore incurs a large memory penalty (see Figure 3). In a second phase the policy reduces its memory usage, which initially drops the success rate, until it learns to balance the three resolution levels correctly and restores a constant 100% success rate while staying under the budget. However, the policy does not reach the working point set by the memory penalty, likely because the penalty for staying under the budget is too weak.

5.3. Evaluation Results

We evaluated the policy on 20 unseen episodes with random point goals reachable within 250 steps, comparing the learned policy against an all-coarse and an all-fine resolution baseline (see Table 1). The learned policy achieves an 80% success rate while using essentially the same memory as the all-coarse baseline (3.9 MB vs. 3.2 MB), which in turn reaches only 65%. Compared to the all-fine baseline, our policy reduces memory usage by 87% on average, at the cost of a moderate drop in success rate.

Even though this policy was trained on a single episode with a single goal, it has essentially learned how to balance resolution and memory given the current score distribution and memory usage. As a result, it generalises to unseen

Table 1. Point-goal navigation on 20 unseen episodes.

	SR $\uparrow$	SPL $\uparrow$	mem (MB) $\downarrow$	steps
learned	0.80	0.601	3.9	78
all_fine	1.00	0.737	31.1	65
all_coarse	0.65	0.478	3.2	106

episodes and goals.

Beyond the aggregate metrics, we visualise where the learned policy spends its memory budget. As shown in Figure 1, the policy concentrates medium resolution where it is needed for the task and keeps the rest coarse, never splitting to fine. Moreover, at this link is possible to see a video of the policy in action: <https://polybox.ethz.ch/index.php/s/sTw7CrngwroY8p7>.

5.4. Discussion

Different tested approaches We tested different approaches for adaptive resolution policies. A first attempt was a policy with per-block discrete actions (SPLIT, KEEP and MERGE): at each step it refined the top- N blocks and coarsened the bottom- N blocks ranked by a task-related score. The policy was trained both with a sparse reward of +1 when the episode was a success and a dense reward on the difference in geodesic distance compared to the ground-truth map (privileged information used only during training). Although this policy eventually learned to overfit a single setup (MP3D, episode 3, point goal), we moved to the threshold setup mainly because to achieve a sufficiently global effect with discrete per-block actions would require a needlessly enormous action space.

Scenes randomization Although the policy generalized to different episodes and goals, training on randomized scenes would lead to a more robust policy on different environments, leading to even better results than the one in Table 1.

Memory budget randomization The policies above were each trained with a single memory budget of 8 MB, and as a result do not generalise to other budgets: the policy effectively ignores the budget entry of the state and applies the same resolution strategy regardless of the available memory. To obtain a single budget-aware policy, we randomize the budget across episodes. This initially proved unstable, as the reward scale varied by an order of magnitude across budgets, preventing the value network from fitting the returns. Clipping the memory penalty to a bounded range removes this instability and yields a budget-dependent behaviour. However, it does not yet reach the requested memory levels precisely, nor does it always succeed at the task, which would likely require careful tuning of the reward weights.

Table 2. Object-goal navigation on 40 HM3D-val episodes.

	SR $\uparrow$	SPL $\uparrow$	Mem (MB) $\downarrow$
all_merge	0.60	0.377	4.0
policy	0.45	0.307	7.1
all_fine	0.65	0.418	48.8

Generalization to different tasks We further questioned whether our policy is able to work for different tasks as well. In PEANUT, object-goal navigation can be split into many point-goal subtasks toward a predicted point goal. However, there are two main additional layers of complexity:

1. the BEV is used by the PSPNet to predict the goal position, so the resolution can affect the quality of the goal prediction.
2. the PEANUT agent also has to declare success when it is close enough to the goal, so the resolution can impact the success of the task with false-positive stops.

In our test the policy performed worse than the all-coarse baseline (see Table 2). We hypothesise that the PSPNet goes out of distribution when fed BEVs at non-uniform resolution, so this test is likely not a fair evaluation of the policy. The next step for a fair evaluation would be to either fine-tune the PSPNet on non-uniform resolution maps, smooth the adaptive map to a more uniform resolution before prediction, or train the policy directly on the object-goal task so that it learns to keep resolution uniform where it is needed to improve goal prediction.

6. Conclusion

We showed that a learning-based approach to goal-driven adaptive resolution is viable: a single budget-aware policy, optimised for the downstream task rather than for reconstruction fidelity, retains navigation performance close to a uniformly fine map while using only a fraction of its memory, and generalises to unseen scenes and goals despite being trained on a single episode.

This opens the route to many promising future directions, such as building a more robust policy through randomization during training and testing our pipeline on agents whose performance is more strongly influenced by the quality of the 3D map.

7. Work Distribution

All authors contributed equally to this work, jointly carrying out the method design, implementation, experimental evaluation, and writing of the manuscript.

References

- [1] Fanfei Chen, Shi Bai, Tixiao Shan, and Brendan Englot. Self-learning exploration and mapping for mobile robots via deep reinforcement learning. In *AIAA SciTech*, 2019. 2
- [2] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011. 1
- [3] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6), 2013.
- [4] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373, 2017. 1
- [5] Alpay Ozkan, Tunc Aydin, Marc Pollefeys, Jelena Trisovic, and Daniel Barath. Unrl: Uncertainty-aware rl-controlled adaptive 3d mapping. *arXiv preprint*, 2026. 2
- [6] Julio A. Placed and Jose A. Castellanos. A deep reinforcement learning approach for active slam. *Applied Sciences*, 2020. 2
- [7] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M. Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. In *NeurIPS Datasets and Benchmarks Track*, 2021. 3
- [8] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A platform for embodied ai research. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. 1, 3
- [9] Lukas Schmid, Jeffrey Delmerico, Johannes L. Schönberger, Juan Nieto, Marc Pollefeys, Roland Siegwart, and Cesar Cadena. Panoptic multi-tdfs: A flexible representation for online multi-resolution volumetric mapping and long-term dynamic scene consistency. In *ICRA*, 2022. 2
- [10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3, 5
- [11] Albert J. Zhai and Shenlong Wang. Peanut: Predicting and navigating to unseen targets. *arXiv preprint arXiv:2212.02497*, 2022. 1, 2, 3
- [12] Jianhao Zheng, Daniel Barath, Marc Pollefeys, and Iro Armeni. Map-adapt: Real-time quality-adaptive semantic 3d maps. *arXiv preprint arXiv:2406.05849*, 2024. 1, 2