Auxiliary Tasks and Exploration Enable ObjectGoal Navigation

Joel Ye[1*] Dhruv Batra[1,2] Abhishek Das[2] Erik Wijmans[1]

[1]Georgia Institute of Technology [2] Facebook AI Research

Abstract

*ObjectGoal Navigation (*ObjectNav*) is an embodied task wherein agents are to navigate to an object instance in an unseen environment. Prior works have shown that end-to-end* ObjectNav *agents that use vanilla visual and recurrent modules,* e.g. *a CNN+RNN, perform poorly due to overfitting and sample inefficiency. This has motivated current state-of-the-art methods to mix analytic and learned components and operate on explicit spatial maps of the environment. We instead re-enable a generic learned agent by adding auxiliary learning tasks and an exploration reward. Our agents achieve* 24.5% *success and* 8.1% *SPL, a 37% and 8% relative improvement over prior state-of-the-art, respectively, on the Habitat ObjectNav Challenge [35]. From our analysis, we propose that agents will act to simplify their visual inputs so as to smooth their RNN dynamics, and that auxiliary tasks reduce overfitting by minimizing effective RNN dimensionality;* i.e. *a performant* ObjectNav *agent that must maintain coherent plans over long horizons does so by learning smooth, low-dimensional recurrent dynamics.*

Site: joel99.github.io/objectnav/

1. Introduction

Consider how a robot placed in a novel home environment should find a target object, *e.g.* 'Find a chair'. This task, known as ObjectGoal Navigation (ObjectNav), requires the agent to search through the unseen environment and, upon seeing a goal, navigate around obstacles to reach it.

Current state-of-the-art ObjectNav agents [5, 35] build explicit spatial maps of the environment and leverage a mix of analytic and learned planning on top of these maps. This is in contrast to the state-of-the-art methods in PointNav, a related task where agents navigate to specified goal coordinates (instead of object categories). In PointNav, the best method [39] scales a generic architecture (*i.e.* a CNN and RNN) to 2.5 billion frames of experience. This approach outpaces methods with explicit spatial grounding, to the point of essentially solving PointNav in Habitat [28].

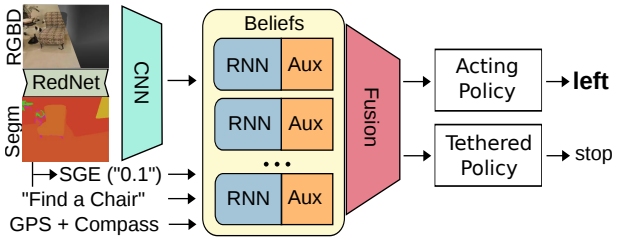

Figure 1. Overview of the ObjectNav agent architecture. The agent receives RGBD input, a GPS+Compass sensor, and must navigate to a goal instance. Building on [40], we introduce new auxiliary tasks, a semantic segmentation visual input, a Semantic Goal Exists (SGE) feature that describes the fraction of the frame occupied by the goal object, and a method for tethering secondary policies that learns from its own reward signal with off-policy updates. We encourage the acting policy to explore, and the tethered policy to perform efficient ObjectNav.

In this work, we explore how to enable such a generic agent for ObjectNav. ObjectNav is considerably more challenging than PointNav; the agent that nearly solves PointNav only achieves 6% success on ObjectNav [35]. While PointNav agents can use the onboard GPS+Compass sensor as a compact representation to measure progress towards the goal, ObjectNav agents cannot; they instead need to be competent at exploration since the target object can be anywhere in the environment. As such, a vanilla CNN+RNN agent tasked with learning these complex representations for ObjectNav is likely to be under-equipped with only an RL reward, even if the reward is shaped to encourage navigation towards a goal [35]. Even with an additional exploration reward and no other feedback to learn better environment representations, efficient exploration is hard [6].

Our approach builds on a recent advance in PointNav [40] that combines simple architectures with auxiliary learning tasks to greatly improve sample complexity in visually complex environments. Specifically, 1) we update the agent's inputs to incorporate egocentric semantic segmentation and a feature that explicitly describes how much of the target object class is in view. 2) We additionally introduce three new auxiliary learning tasks: two general task for learning

*Correspondence to joel.ye@gatech.edu

an inverse dynamics model and one exploration-centric task that predicts map coverage, and 3) to teach the agent to explore, we add an exploration term to the reward.

However, exploration is different than OBJECTNAV and we find that agents trained with an exploration reward will continue to explore after locating goal instances. To mitigate this task mis-alignment, we propose a 'tethered-policy' multitask learning technique. In this method, we learn a second policy adapted only for the sparse OBJECTNAV reward while still acting via a primary policy given the exploration reward. Tethering allows faster adaptation to the sparse OBJECTNAV reward, which in turns improves agent efficiency over simple fine-tuning of a single policy on the sparse reward.

Our results show that end-to-end learning can achieve state-of-the-art results in OBJECTNAV once equipped with generic representation learning and exploration objectives. Nonetheless we are far from human-like competence at the task. To guide future work, we analyze various aspects of our agent, but center in particular around its *recurrent dynamics*. We propose that zero-shot transfer to RedNet segmentation is greatly degraded by the chaotic dynamics it causes in agent RNNs, that several failure modes are caused by lapses in agent memory, and that auxiliary tasks reduce overfitting by constraining the effective dimensionality of agent RNNs.

Concretely, our contributions are:

1) Objectives to train a simple CNN+RNN architecture that lead to 24.5% success on OBJECTNAV (+37% relative improvement over prior state-of-the-art). This suggests that, despite their current prevalence, explicit maps need not be necessary to learn complex Embodied AI tasks.
2) A method for tethering secondary policies to an agent. Tethered policies learn from separate reward signals with off-policy updates. We use a tethered policy with no exploration reward to achieve 8.1% SPL (+8% relative improvement over prior work).
3) An analysis of the agent. Through examination of the agent's behavior, representations, and recurrent dynamics, we find a) the agent seeks out simple visual inputs corresponding to smoother RNN dynamics, b) agent failures are often related to agent memory, and c) auxiliary tasks regularize agent RNNs by constraining their effective dimensionality. We hypothesize a key component of performant OBJECTNAV agents will be the ability to plan with smooth, low-dimensional recurrent dynamics.

2. Approach

OBJECTNAV Definition and Dataset. [2] In OBJECTNAV, an agent must navigate to an instance of a specified object category in an unseen environment. The agent does not receive a map of the environment and must navigate using its (noiseless) onboard sensors: an RGBD camera, a GPS+Compass sensor which provides location and orientation relative to start of episode. The agent also receives a goal object category ID. The full action space is discrete and consists of MOVE_FORWARD, TURN_LEFT, TURN_RIGHT, LOOK_UP, LOOK_DOWN, and STOP. We experiment with both the full 6-action setting and a restricted 4-action setting that excludes LOOK_UP and LOOK_DOWN. To our knowledge, no other works have considered the 6-action setting (perhaps due to the increased complexity of specifying tilt actions in 2D map-based planners). The agent must stop at a location within $1m$ of an object of the specified category and be able to view the object from that location. There is a 500 step limit to succeed on the episode.

We experiment on the Matterport dataset (MP3D [4]), which has 90 scenes and 40 labeled semantic object categories. There are 21 goal categories, as specified in the Habitat 2020 Challenge [35]. We train our agent with a 3D GPS (*i.e.* with vertical localization) to encourage the agent to reason about 3D exploration, as a considerable number of MP3D scenes incorporate elevation changes. However, we evaluate the agent with a 2D GPS for compatibility (by zero-ing vertical axis) with the Habitat Challenge parameters. We provide a comparison of these settings in (Section A.4).

Additional Features. We accelerate agent learning by providing two semantic features: semantic segmentation (segm.) for the visual input, and a "Semantic Goal Exists" (SGE) scalar which equals the fraction of the visual input that is occupied by the goal category (computed from the semantic segm.). During training, we use the ground truth semantic segm. directly from the MP3D annotations. During evaluation, the segm. is predicted from RGBD using a RedNet [18] model finetuned to predict the 21 goal categories. The RedNet model is taken off-the-shelf (trained on SUNRGBD [32]) and finetuned on 100k randomly sampled forward-facing views from MP3D. We provide additional details in Section A.15. The SGE feature distills the domain knowledge that the agent should navigate to the goal once it is seen; this knowledge is built into the planners in prior works [5, 20].

2.1. Agent Architecture

Our agent uses the split-belief architecture introduced in [40], shown in Fig. 1. This approach first embeds all sensory inputs with feed-forward modules. The visual inputs are first downsampled by 0.5 *i.e.* from 480×640 to 240×320, and the semantic segm. channel is also projected to 4D by associating each semantic ID with a learnable vector. After this preprocessing, visual inputs are fed through a ResNet18 [15]. The goal object ID is directly embedded into a 32D vector. The embedded visual and goal vectors are concatenated with the GPS-Compass and SGE inputs to form an observation embedding. Next, a set of recurrent "belief modules" integrate these observation embeddings over time. These belief modules are independent GRUs [7], each associated with a separate auxiliary task; Ye *et al.* [40]

proposes that the split modules, as opposed to one larger recurrent network, enable orthogonal auxiliary tasks to be learned while minimizing task conflict. We term output cell states from all GRUs as "beliefs." Beliefs are fused using an attention layer conditioned on the observation embedding. The fused belief is then directly used in a linear actor-critic policy head. We refer to this agent as the base agent.

2.2. Learning Signals

Rewards. The base agent receives a sparse success reward r_{success}, a slack reward r_{slack} to encourage faster goal-seeking, and an exploration reward r_{explore}. Measuring exploration is a dense indicator of progress, and encourages an intuitively prerequisite skill for OBJECTNAV. We use a visitation-based coverage reward, in which we first divide the map into a voxel grid with $2.5m \times 2.5m \times 2.5m$ voxels and reward the agent for visiting each voxel. This form of visitation-based exploration bonus was found to work well for map-based agents [26]. We smooth r_{explore} by decaying it by the number of steps the agent has spent in the voxel (visit count v). Further, to ensure the agent eventually prioritizes OBJECTNAV, we decay r_{explore} based on episode timestep t with a decay constant $d = 0.995$. In summary:

$$r_{\text{total}} = r_{\text{success}} + r_{\text{slack}} + r_{\text{explore}} \tag{1a}$$

$$r_{\text{success}} = 2.5 \quad \text{on success} \tag{1b}$$

$$r_{\text{slack}} = -10^{-4} \quad \text{per step} \tag{1c}$$

$$r_{\text{explore}} = 0.25 \times \frac{d^t}{v} \tag{1d}$$

Tethering to an Exploration Policy. Using exploration to guide behavior can distract the agent from properly terminating at the goal point, *e.g.* when it is standing next to the goal but can instead choose to continue to explore a wide open space (we observe this in Section 4.1). To alleviate this pathology, we introduce a tethered-policy method which attaches an additional policy head to the fused belief output. Each of the two policy heads can be fed their own desired rewards, and the agent can act according to any mix of the two policies. Note this means that each policy share the same action space. We implement a simple schedule in which the agent first acts with a policy optimized with the full shaped reward (r_{total}) for an initial period of training, and then with a second policy which only uses the sparse r_{success}. The first phase is intended to teach the agent how to explore, and the second to encourage efficient OBJECTNAV. We use importance-weighted VTrace [10] returns to update the non-behavioral policy to account for experience being off-policy from its perspective (details in Section A.13). Tethering is similarly motivated as SAC-X [27], but implemented as a lightweight extension to a standard RL agent.

Auxiliary Tasks. We use 6 auxiliary tasks. As in Ye *et al*. [40], we use 2 instances of CPC|A [14], a self-supervised contrastive task, at horizons of $k = \{4, 16\}$ steps. We also use PBL [13] with a horizon of $k = 8$ steps. These tasks use agent states to make predictions about the environment.

We also introduce two general-purpose inverse tasks, which predict agent actions given environment transitions: Action Distribution Prediction (ADP) and Generalized Inverse Dynamics (GID). ADP and GID both predict actions taken between two observations k frames apart (*i.e.* $\{a_{[t:t+k]}\}$), and are both conditioned on the belief at the first frame h_t and the visual embedding (*i.e.* CNN output) ϕ_{t+k}. ADP uses a 2-layer MLP to directly predict an action distribution and evaluates the KL-divergence between this prediction and the empirical distribution of the next k actions:

$$L_{\text{ADP}} = KL(\text{MLP}(h_t, \phi_{t+k}), a_{[t:t+k]}) \tag{2}$$

GID predicts individual actions using a separate GRU with state g^{GID} and two linear layers f, f'. The GRU is initialized with the same inputs as in ADP:

$$g_t^{\text{GID}} = f(h_t, \phi_{t+k}) \tag{3}$$

The GRU is updated with actions taken:

$$g_{t+i}^{\text{GID}} = \text{GRU}(a_{t+i-1}, g_{t+i-1}^{\text{GID}}) \tag{4}$$

$$L_{\text{GID}} = \sum_{i=1}^{k-1} \text{CrossEnt}(f'(g_{t+i}^{\text{GID}}), a_{t+i}) \tag{5}$$

Finally, we introduce an exploration-specific Coverage Prediction (CP) task that leverages the GPS sensor provided to the agent. CP follows a similar structure as GID, *i.e.* using a GRU for sequence prediction. However, this GRU's initial state is conditioned not on a visual embedding, but on the number of steps the agent has spent in its current voxel, $v(s_t)$. This conditioning helps the GRU anticipate how close the agent is to the edge of the voxel. The GRU predicts the change in coverage at each of the next k steps:

$$g_t^{\text{CP}} = f''(h_t, v(s_t)) \tag{6}$$

$$g_{t+i}^{\text{CP}} = \text{GRU}(a_{t+i-1}, g_{t+i-1}^{\text{CP}}) \tag{7}$$

$$L_{\text{CP}} = \sum_{i=1}^{k} \text{CrossEnt}(f'''(g_{t+i}^{\text{CP}}), \Delta_{\text{cov}}(t, t+i)) \tag{8}$$

where f'', f''' are linear layers, g^{CP} denote the GRU's state, and $\Delta_{\text{cov}}(t, t+i)$ denotes the change in coverage, *i.e.* number of new voxels visited between time t and $t + i$[1]. Perfect performance would require the belief to remember all prior locations. The horizons for ADP, GID, and CP are $k = 6, 4, 16$, respectively; details in Section A.14.

[1]Note that $\Delta_{\text{cov}}(t, t+i) \in \{0, \ldots, i\}$

	VAL		TEST-STD	
	Success % (↑)	SPL % (↑)	Success % (↑)	SPL % (↑)
1) 4-Action	**34.4±2.0**	9.58±0.75	19.9	6.7
2) 4-Action + RedNet FT	**33.1±2.0**	6.89±0.56	-	-
3) 6-Action	30.8±1.9	7.60±0.64	-	-
4) 6-Action + Rednet FT	**34.6±2.0**	7.93±0.64	**24.5**	6.4
5) 6-Action + Tether	26.6±1.9	9.79±0.82	-	-
6) 6-Action + Tether + RedNet FT	30.3±1.9	**10.8±0.84**	21.1	**8.1**
7) E2E Baseline (DD-PPO [39])	-	-	6.2	2.1
8) SemExp [5]	-	-	17.9	7.1
9) SRCB-robot-sudoer	-	-	14.4	7.5

Table 1. Primary variants on the MP3D VAL and TEST-STD splits, along with prior SoTA from the Habitat Challenge leaderboard [35]. ± intervals provide 95% CI; VAL bold values are significantly better than non-bold values ($p < 0.05$, paired t-test), TEST bolding is for emphasis. Our auxiliary task and exploration enabled agent (row 1) outperforms prior state of the art success (over row 7) and triples the vanilla learning agent (row 6). Fine-tuning the agent with RedNet segmentation reverses the ranks of 4 and 6-action agents (rows 1 and 3 vs 2 and 4, analysis in Section 3.1), helping the 6-action agent to achieve a new state-of-the-art 24.5% success. Finally, tuning on the sparse OBJECTNAV success reward helps the agent set state-of-the-art 8.1% SPL (row 6 vs 9).

3. Results

In our experiments, we train each of our agents for 8 GPU-weeks (192 GPU-hours), amounting to ∼125M frames per agent. For the tethered variant, we use the sparse policy for the final 25M frames. The two primary metrics we report are Success and Success weighted by Path Length (SPL), a measure of trajectory efficiency, defined over N episodes as

$$\text{SPL} = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)}$$

where S_i is a binary indicator of success on episode i, l_i is length of optimal path, and p_i is length of agent path. We evaluate checkpoints every ∼4M frames and report metrics from the checkpoint with the highest SPL on VAL.

Auxiliary Tasks and Exploration produce an effective OBJECTNAV agent. We compare against results from the Habitat Challenge 2020 leaderboard in Table 1. We first note the large drop from VAL to TEST-STD (*e.g.*-14% success in row 1). Since we cannot access the test split, we conjecture that the splits were randomly sampled such that the 9 test scenes were disproportionately challenging.[2] While other works do not highlight this shift, we note that even agents with strong priors (rows 6, 7) have high performance variance, *e.g.* 5% success gaps between the two test splits (TEST-STD, TEST-CHALLENGE [35]). We provide additional commentary and check for agent biases in Section A.1.

[2]Distribution shift is expected given small VAL, TEST splits.

Using RedNet segm. directly (*i.e.* zero-shot transfer from GT , row 1), the 4-action agent reaches 19.9% success (+11% relative over prior best, row 1 vs 8). This is 3x the performance of the E2E baseline that is rewarded for approaching a goal, a method which "solved" POINTNAV (row 7).

We tune the 4-action agent with RedNet segm. for an additional 10M frames to account for the ground-truth-to-RedNet distribution shift, but surprisingly performance *drops* (row 1 vs 2). Conversely, the 6-action agent which initially performs worse than the 4-action agent (row 1 vs 3) overtakes the 4-action agent after finetuning with RedNet (row 2 vs 4). We analyze this reversal in Section 3.1. Though these tuned agents match in VAL (row 1 vs 4), the 6-action agent improves on TEST-STD, to 24.5%, +37% relative over prior state-of-the-art (row 4 vs 8).

However, both 4 and 6-Action agents are less efficient than prior methods (SPL column). Qualitative examination of these agents indicate they prefer to wander around goals for some time before stopping at them correctly, likely due to the exploration reward. After finetuning on the sparse OBJECTNAV reward through tethered training, SPL rises and Success falls (row 3 vs 5). We show the Success drop is likely due to reduced exploration in Section A.2. Nonetheless, after tuning this agent on RedNet, we achieve +0.6% (+8% relative) over state of the art SPL (row 6 vs 9).

We provide ablations for our design choices in Table 2. Our auxiliary task ablations are compared against the 4-action agent (rows 1-4). These tasks provide modest gains in both metrics (<1% for ADP, 1% GID, 2% CP). Next, ablating the

	Success % (↑)	SPL % (↑)
1) Base (4-Action)	**34.4**±2.0	**9.58**±0.75
2) - Action Distribution Prediction	**34.1**±2.0	**9.27**±0.69
3) - Generalized Inverse Dynamics	**33.7**±2.0	**9.05**±0.69
4) - Coverage Prediction	32.4±2.0	8.7s7±0.69
5) - Semantic Goal Exists	20.3±1.7	4.14±0.47
6) 6-Action + Tether	**26.6**±1.9	**9.79**±0.82
7) 6-Action + Sparse Tuning	23.1±1.8	8.43±0.76

Table 2. Ablations on VAL. Bold values are significantly better than unbolded values in the same group. New auxiliary tasks provide moderate performance gains, and tethered training preserves Success better than directly tuning on sparse OBJECTNAV reward.

	Success % (↑)		SPL % (↑)	
	TRAIN	VAL	TRAIN	VAL
1) 4-Act	50.3 (36.0)	43.3 (**34.4**)	18.1 (12.4)	12.3 (9.6)
2) 6-Act	56.0 (21.7)	**58.0** (30.8)	21.5 (8.2)	16.9 (7.6)
3) 6-Act + Tether	54.0 (27.3)	52.2 (26.0)	27.9 (11.5)	**26.0** (**9.8**)

Table 3. Performance on a 300-episode subset of TRAIN and VAL splits, reported as "with GT segmentation (with RedNet segmentation)". Best metrics are bolded for emphasis. In both splits, all agents degrade significantly with RedNet segmentations, but 6-action agents more so. Under GT segmentation (numbers outside parentheses), agents are minimally overfit, though SPL does degrade slightly from TRAIN to VAL. In the GT setting, 6-action agents outperform 4-action agents.

SGE sensor drops performance by a large amount (-14%, row 1 vs 5). In Section A.5, we note that SGE is much more effectively incorporated as a feature than as an auxiliary task. Finally, we also compare tethered policy training against direct finetuning (row 6 vs 7) for the 6-action agent. While direct finetuning also improves SPL (vs SPL reported in row 3 in Table 1), tethered training is overall more effective at preserving agent performance.

3.1. Stable segmentation is critical to untuned 6-action agents.

We found that 4-action agents suffer a performance drop after finetuning on RedNet, while 6-action agents improve. To understand why, we first observed that the RedNet is greatly overfit (Section A.15), we decouple our agent from RedNet by evaluating agent performance on both TRAIN and VAL with GT and RedNet segm. , shown in Table 3. We first note that most agents are minimally overfit to TRAIN, with a moderate impact to SPL and $< 3\%$ drop in success. RedNet Seg. degrades performance significantly, but in particular, RedNet Seg. hurts 6-action agents more, even on TRAIN, where the effects of an overfit RedNet are reduced. This suggests 6-action agents are more sensitive to RedNet statistics in particular, and hence able to overtake the 4-action agent once finetuned with RedNet.

We examine agent trajectories to understand this sensitivity (videos provided in supplement). We first find that during training with GT semantics, 6-action agents learn to navigate while spending most steps facing downwards towards the floor (while the 4-action agent can only look straight ahead). This happens despite there being minimal obstacles on the floor while looking straightforward offers greater chance to see more distant goals. Intuitively, the visual input provided by the floor is simpler than the view of a room. We thus posit a stability hypothesis: the 6-action agent learns to exploit a *simpler signal to enable stabler recurrent dynamics*. That is, because it is harder to maintain memory over long term trajectories when faced with noisy inputs, agents will simplify their input even at slight cost to information gained. Such a hypothesis may relate to results on improved generalization when restricting the visual field of a gridworld agent [17] and when withholding inputs from parts of a recurrent module [12]. This hypothesis suggests that standard RNN gating is imperfect at preventing input-induced information decay in the RNN state, and presents an avenue for future work.

We next observe that RedNet segm. is low quality and unstable, flickering across consecutive frames and without well-defined object boundaries. These are expected failures: poor segm. is due in part to the noisiness of the underlying MP3D meshes, and consistent segm. is a broader challenge [36]. Predicted segm. of frames of the floor are particularly poor, likely since RedNet is trained with front-facing views. Consequently, the 6-action agent behavior was very erratic, with repeated alternating turning actions. If the 6-action agent did learn to exploit the floor for simple visual inputs, it would have a reduced incentive to filter noisy frontal views, and thus would be particularly vulnerable to degraded segm. Then, one possible reason to explain earlier trends is that 1. RedNet overfitting implies that previously general agents which are tuned with RedNet will in turn overfit, 2. Untuned 6-action agents avoid this overfit transfer but suffer significantly more from unstable semantics, and 3. Tuned 6-action agents overfit, but gain performance by maintaining stable behavior while receiving unstable semantics. We quantify this instability in Section A.7. The fact that tuning agents with RedNet improves agent performance suggests robustness to noisy inputs is partially learnable, though the agent will not do so if it has a simpler option (facing the ground).

4. Agent Analysis

These agents demonstrate promising performance ($\sim 55\%$ val success) in navigating large, complex environments, especially when granted GT segm. Nonetheless, their performance is far from perfect, and they still display qualitative failure modes that a human would not have, such as repeated circling inside a room. In this section we analyze agent be-

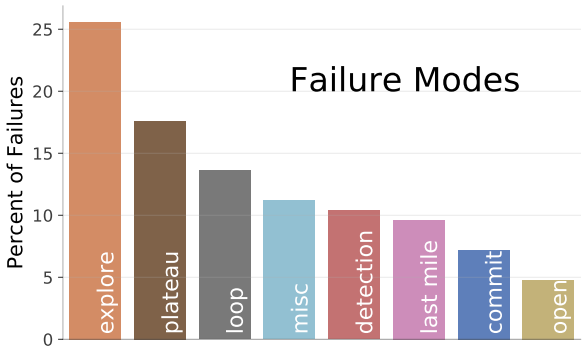

Figure 2. Breakdown of failure modes for base 6-action agent in 300 episodes of VAL.

havioral modes, knowledge, and internal dynamics to inform future directions towards human-level OBJECTNAV, and to develop an intuition for how a competent OBJECTNAV agent operates. We scope our analysis to GT segm. and the base 6-action agent, unless otherwise noted.

4.1. Behavioral Analysis

In our behavioral analysis we aim to understand: How can OBJECTNAV agents approach 1.0 Success? To identify prominent modes of the remaining failures, we conduct a qualitative coding of agent behavior for both the base and tethered 6-action agent. Specifically, we sample 300 validation episodes and manually label the failures (there are 125 for base, 151 for tether) and group them according to common trends. A subset of failure modes are described in Table 4 and their relative prevalence for the base agent is shown in Fig. 2. A full list of failure modes are in Section A.6, and example videos are available in the supplement.

Name	Description
Explore	A generic failure to find the goal despite steady exploration. Includes semantic failures *e.g.* going outdoors to find a bed.
Plateau	Repeated collisions against the same piece of debris causes a plateau in coverage. Includes debris which traps agent in spawn.
Loop	Poor exploration due to looping over the same locations or backtracking.
Detection	Despite positive SGE, the agent does not notice nor successfully navigate to the goal.
Last mile	Gets stuck near the goal.
Commitment	Sees and approaches the goal but passes it.
Open	Explores an open area without any objects.

Table 4. Description of prominent failure modes.

We find that agent failures are diverse and beyond only unsuccessful exploration, which at best comprises 'explore' and 'loop'. They get stuck ('plateau' and 'last mile'), suffer pathologies that might be attributed to the exploration reward ('commitment' and 'open'), and deal with a handful of issues due to scene quality (labeled as 'misc' here). However, some failure modes do preclude others. For *e.g.*, an agent which is trapped by reconstruction debris at spawn ('plateau') is still liable to not find the goal due to general exploration issues. Additionally, the agent does not make panoramic turns to identify promising goal locations, as a human might. We attribute this to the visitation-based exploration reward; a different reward which promotes simply viewing more areas instead of visiting them may mitigate this. Alternately, non-episodic exploration rewards may promote the appropriate behavior with less misalignment with OBJECTNAV.

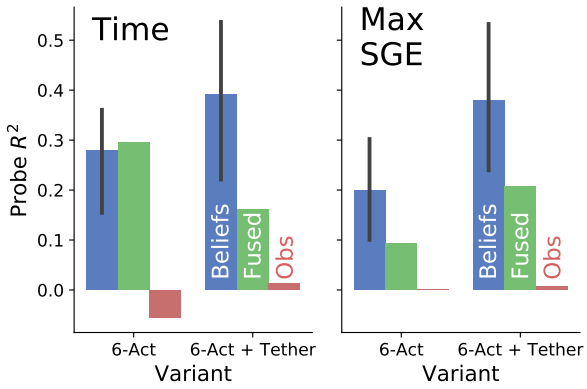

Figure 3. We probe time and max SGE seen on each belief, the fused belief, and the observation embedding ("Obs") as a baseline. Error bars show 95% CI across single beliefs. Both concepts are encoded, though imprecisely and with high variance across beliefs.

These exploration-related pathologies are not cured by swapping to the sparse OBJECTNAV reward, which degrades success (Table 2, row 6 and 7). Tethered agents explore worse, and behavioral coding finds a new dominant failure mode wherein the agent quits early (see Section A.2).

4.2. Probing Learned Knowledge

How well do agent representations match our intuitions? Does its behavior match represented features? *e.g.*, "commitment" failures, which generally occur when the agent spots the goal early in the episode, should only be feasible if the agent maintains a time representation and knows it has time to continue gaining coverage rewards. Conversely, does the agent represent prominent variables that a human might? *e.g.*, "Commitment" and "Detection" failures both imply that agents do not remember SGE features from prior timesteps. To test these questions, we train linear decoders to probe our agent for time and the most goal seen. We first record representations from 300 episodes on both TRAIN and VAL. We train the decoders using TRAIN representations and test them on VAL representations ($\sim 60 - 70$K steps each).

We plot decoder performance in Fig. 3, aggregating performance across individual beliefs. Though the features are represented across beliefs, there is high variance in probe performance across beliefs. Moreover, probe performance on the fused belief is worse than in individual beliefs. These results reveal semantic differences among individual beliefs and the fused belief, suggesting some abstraction is lost in fusion despite it only being a weighted sum. Separately, the tethered agent represents Max SGE more prominently than the base agent, matching the intuition that the base agent does not retain goals seen in previous steps perfectly.

With the same approach, we find little evidence of features that a human might use in ObjectNav: time spent in location, room ID, and distance to goal (Section A.8). Supplying these features directly could improve the agent.

4.3. Dynamical Analysis

Though a behavioral analysis provides a sense of how the agent fails, it does not explain why, and probing techniques fall short when it is unclear what knowledge an agent might encode. Through our experiments, dynamics have played an important role – the 6-action agent looking downward, acting chaotic with predicted segm., and overtaking 4-action after tuning. Important failure modes appear dynamic in nature: 'loop' failures indicate forgetting of visited locations, 'commitment' failures could be attributed to procrastination through the agent's internal clock. We were thus motivated to characterize the agent's dynamic computations.

Belief Dynamics correlate with agent observations. Before we study RNN dynamics proper, we first validate that RNN dynamics reflect trajectory dynamics, as represented by sequences of observation embeddings. This helps quantify to what extent agent beliefs are grounded in agent observations and actions; a strong connection would support a stability hypothesis (that agents need smooth inputs for smooth RNN dynamics). Alternately, RNN dynamics could be driven by intermediate computation, and thus the RNNs would conservatively incorporate inputs, *i.e.* through gating, so as to best preserve hidden state information. We measure this with the curvature of the agent belief trajectories and the observation embeddings, by computing the dot product similarity of successive displacements in a sequence of representations ([16], details in Section A.9). We report the correlation between each belief curvature and observation curvature.

Fig. 4 shows that observation and belief curvatures are closely correlated, *i.e.* that varying observations track large changes within agent hidden state. This suggests that RNNs do incorporate most inputs. It is possible non-reactive computation (*e.g.* such as time tracking probed in Section 4.2) which occurs independently from observations might not manifest because they are relatively stable; however, the CP belief, which should track coverage variables that are relatively independent from moment-to-moment observations, is

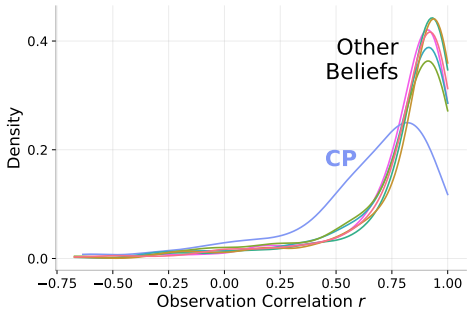

Figure 4. We measure curvature of observation embeddings and beliefs of the base 6-action agent, plotting the distribution from transitions in 300 episodes. All but CP closely correlate.

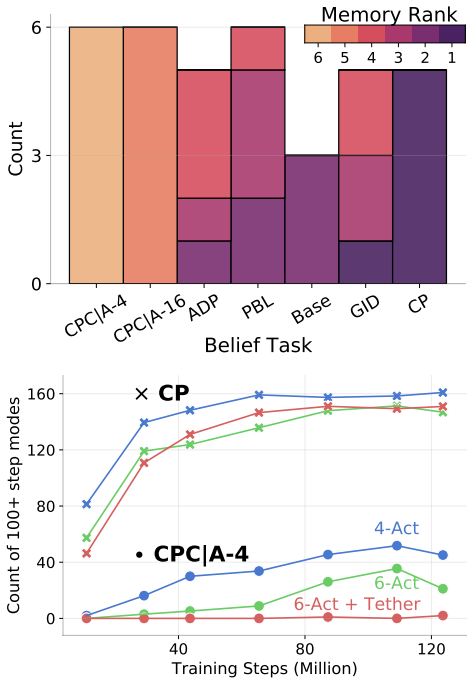

Figure 5. Fixed point (FP) memories. Top: For each of 6 agents (4-Act, 6-Act, 6-Act + Tether, No CP, No ADP, No GID), we sort agent beliefs by the span of their memory. This span is computed by sampling 50 FPs and counting at each FP the number of eigenvalues with time constants above 10 steps. The more eigenvalues past threshold, the larger the span, and the lower the rank. Beliefs are labeled by their task, and beliefs without tasks (from ablations) are labeled with Base. We count the number of times a belief appears in a given rank (*e.g.* since Base only appears in 3 agents, it only has 3 counts). Consistent rank indicates *relative span is stable.* Bottom: We measure how memory changes over time for the CPC|A-4 and CP tasks by counting the number of eigenvalues with time constants above 100 steps at several points during training.

less correlated with observations than the other beliefs. This suggests non-reactive computation needn't be hidden.

Auxiliary tasks specify fixed point memory spans. We would ideally like to understand what computational structure the agent RNNs use beyond coarse-grained statistics like curvature. To approach this question, we employ fixed point

(FP) analysis [11, 33, 34]. FP analysis simplifies the study of a full RNN into a study of its "slow points," where the RNN update is small and well-approximated by a linear function. We provide details on methodology in Section A.10. FPs are typically studied both through their overall layout (the manifold) and their local dynamics. While FP manifolds have been used to directly link RNN computation to dynamical structures, *e.g*. ring attractors, we find our RNNs are much higher-dimensional than those in prior work and thus difficult to clearly classify (notes in Section A.11). This leaves local dynamic structure. At an identified FP h, the RNN update is well described by a first order approximation. Let $\Delta h_t := h_t - h$, for h_t near h, then

$$\Delta h_{t+1} \approx J_h^{\text{rec}} \Delta h_t. \tag{9}$$

Then the decomposition of J_h^{rec} assesses FP "memory." Specifically, a delta along an eigenvector v with corresponding eigenvalue $\lambda \ll 1$ would decay to 0, while an eigenvector v' with $\lambda' \approx 1$ would persist indefinitely. A simple method to quantify FP memory is to count the eigenvalues past a threshold value. For interpretability we count eigenvalue time constants instead of the values themselves, as in [21] (Section A.10); we call this quantity a memory span.

We measure and rank these spans for all beliefs of several different agents, producing Fig. 5. The auxiliary task corresponding to a belief appears to play a consistent role in specifying FP memory span, and thus in organizing recurrent dynamics, across seeds, action suites (4 vs 6 action), and RL objectives (tether or not). Among the tasks, only CP induces larger spans than the base belief. This matches the intuition that CP could require the tracking of an unbounded number of variables. Separately, the fact that the base belief has larger span than other tasks reveals a potential mechanism by which auxiliary tasks encourage generalization. Note that smaller memories imply there are more dimensions along which RNN state decays towards fixed points, which in turn should imply preference towards lower-dimensional trajectories (even if variable inputs prevent this empirically). Then, the fact that most auxiliary task-augmented beliefs achieve smaller memories than the vanilla "Base" belief demonstrates auxiliary tasks may preferentially select for low-dimensional trajectories. Such low-dimensional trajectories could be necessary for generalization; indeed this is theorized to be the reason why monkeys engage low-d dynamics in working memory tasks[8]. Note that this hypothesis concerns how auxiliary tasks promote generalization in *recurrent* beliefs (*i.e*. beyond general visual representations).

In the bottom figure of Fig. 5 we plot memory span for several agents at various points in training. Memory span trends upward over training. Consistent with the first plot, CPC|A-4 appears to constrain the span to be low. CP, on the other hand, may be constrained by the capacity of the RNN (the belief hidden size is 196). It is possible that an ideal OBJECTNAV agent would maintain stable low-d dynamics, but this would need to be reconciled with how such an agent would need to track its path to keep exploring new areas.

5. Related Work

Learning an appropriate state representation is a central goal of reinforcement learning. Representations pretrained in static visual tasks, such as the semantic segm. used in our agent, have been successfully applied to navigation in complex environments [24, 29], though results in [40] suggest non-semantic representations may be less effective than end-to-end learned representations. While auxiliary tasks are commonly used to improve agent performance [13, 23], we believe their role in preventing overfitting to complex environments is underappreciated.

Analyzing Embodied Recurrent Computation. Understanding an agent's recurrent computation will be key to understanding their complex behavior. While complex computation resists simple analyses, *e.g*. our probes, the AI community has successfully used sophisticated probing tasks to reveal recurrent representations reflect considerably more environmental knowledge than do static (visual) representations [37], and can contain compositional knowledge [9]. We have explored a different perspective by analyzing the dynamic rules our agent learns; similar analyses have provided considerable insight in simpler tasks [1, 33]. Unfortunately, our agent RNNs are much higher-dimensional than those studied in previous settings and stymies simple fixed point analyses. However, we are encouraged by the different perspective (*e.g*. memory) such dynamic analyses provide, and believe embodied AI provides a good testbed for future development of these dynamic analyses.

6. Discussion

Our work uses a recurrent agent with an implicit environment representation to set state-of-the-art for OBJECTNAV. Our agents are minimally overfit and have yet to saturate training performance, meaning that our approach re-enables scaling as a viable path to improving OBJECTNAV. This success relies on auxiliary tasks and an exploration policy, both of which are generic RL ingredients. Future works should consider these ingredients in their generic CNN+RNN baselines, since the default of only providing rewards is demonstrably ineffective in complex environments. Our analysis indicates that understanding misbehaving agent dynamics is a promising direction, and our results suggest generic learned agents can still drive progress in embodied navigation.

Acknowledgments. The authors thank Vincent Cartillier for help preparing the RedNet models. EW is supported in part by an ARCS fellowship. The Georgia Tech effort was supported in part by NSF, AFRL, DARPA, ONR YIPs, ARO PECASE, Amazon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government, or any sponsor.

References

[1] Kyle Aitken, Vinay V. Ramasesh, Ankush Garg, Yuan Cao, David Sussillo, and Niru Maheswaranathan. The geometry of integration in text classification rnns, 2020. 8

[2] Peter Anderson, Angel X. Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Roshan Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 2

[3] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic mapnet: Building allocentric semanticmaps and representations from egocentric views, 2020. 16

[4] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017. MatterPort3D dataset license available at: http://kaldir.vc.in.tum.de/matterport/MP_TOS.pdf. 2

[5] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *In Neural Information Processing Systems*, 2020. 1, 2, 4

[6] Devendra Singh Chaplot, Saurabh Gupta, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural mapping. In *ICLR*, 2020. 1

[7] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*, 2014. 2

[8] Christopher J. Cueva, Alex Saez, Encarni Marcos, Aldo Genovesio, Mehrdad Jazayeri, Ranulfo Romo, C. Daniel Salzman, Michael N. Shadlen, and Stefano Fusi. Low-dimensional dynamics for working memory and time encoding. *Proceedings of the National Academy of Sciences*, 117(37):23021–23032, 2020. 8

[9] Abhishek Das, Federico Carnevale, Hamza Merzic, Laura Rimell, Rosalia Schneider, Josh Abramson, Alden Hung, Arun Ahuja, Stephen Clark, Gregory Wayne, and Felix Hill. Probing emergent semantics in predictive agents via question answering. In *International Conference on Machine Learning*, 2020. 8

[10] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. 2018. 3, 15

[11] Matthew D. Golub and David Sussillo. Fixedpointfinder: A tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. *Journal of Open Source Software*, 3(31):1003, 2018. 8

[12] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard SchÃ¶lkopf. Recurrent independent mechanisms, 2020. 5

[13] Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning, 2020. 3, 8

[14] Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A Pires, Toby Pohlen, and Rémi Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018. 3

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 2

[16] Olivier J Hénaff, Robbe LT Goris, and Eero P Simoncelli. Perceptual straightening of natural videos. *Nature neuroscience*, 22(6):984–991, 2019. 7, 14

[17] Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L. McClelland, and Adam Santoro. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*, 2020. 5

[18] Jindong Jiang, Lunan Zheng, Fei Luo, and Zhijun Zhang. Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation, 2018. 2, 16

[19] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 16

[20] Yiqing Liang, Boyuan Chen, and Shuran Song. Sscnav: Confidence-aware semantic scene completion for visual semantic navigation, 2020. 2

[21] Niru Maheswaranathan and David Sussillo. How recurrent networks implement contextual processing in sentiment analysis. *arXiv preprint arXiv:2004.08013*, 2020. 8, 14

[22] Niru Maheswaranathan, Alex Williams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics, 2019. 14

[23] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. *CoRR*, abs/1611.03673, 2016. 8

[24] A. Mousavian, A.toshev, M. Fiser, J. Kosecka, A. Wahid, and J. Davidson. Visual representations for semantic target driven navigation. In *International Conference on Robotics and Automation (ICRA)*, 2019. 8

[25] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 16

[26] Santhosh K. Ramakrishnan, Dinesh Jayaraman, and Kristen Grauman. An exploration of embodied visual exploration, 2020. 3

[27] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Jost Tobias Springenberg. Learning by playing - solving sparse reward tasks from scratch, 2018. 3

[28] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied AI research. In *ICCV*, 2019. 1

[29] Alexander Sax, Bradley Emi, Amir R Zamir, Leonidas Guibas, Silvio Savarese, and Jitendra Malik. Mid-level visual representations improve generalization and sample efficiency for learning visuomotor policies. *arXiv preprint arXiv:1812.11971*, 2018. 8

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016. 16

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 16

[32] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015. 2

[33] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013. 8

[34] David Sussillo, Mark M Churchland, Matthew T Kaufman, and Krishna V Shenoy. A neural network that finds a naturalistic solution for the production of muscle activity. *Nature neuroscience*, 18(7):1025–1033, July 2015. 8

[35] Habitat Team. Habitat challenge, 2020. https://aihabitat.org/challenge/2020, 2020. 1, 2, 4

[36] Serin Varghese, Yasin Bayzidi, Andreas Bar, Nikhil Kapoor, Sounak Lahiri, Jan David Schneider, Nico M Schmidt, Peter Schlicht, Fabian Huger, and Tim Fingscheidt. Unsupervised temporal consistency metric for video segmentation in highly-automated driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 336–337, 2020. 5

[37] Luca Weihs, Aniruddha Kembhavi, Kiana Ehsani, Sarah M Pratt, Winson Han, Alvaro Herrasti, Eric Kolve, Dustin Schwenk, Roozbeh Mottaghi, and Ali Farhadi. Learning generalizable visual representations via interactive gameplay. In *International Conference on Learning Representations*, 2021. 8

[38] Erik Wijmans, Irfan Essa, and Dhruv Batra. How to train pointgoal navigation agents on a (sample and compute) budget, 2020. 16

[39] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, 2020. 1, 4

[40] Joel Ye, Dhruv Batra, Erik Wijmans, and Abhishek Das. Auxiliary tasks speed up learning pointgoal navigation. In *CoRL*, 2020. 1, 2, 3, 8, 16