On-policy Object Goal Navigation with Exploration Bonuses

Eric Maia

Thesis submitted in partial fulfillment of the requirements for

Master of Applied Science

in

Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Machine learning developments have contributed to overcome a wide range of issues, including robotic motion, autonomous navigation, and natural language processing. Of note are the advancements of reinforcement learning in the area of object goal navigation — the task of autonomously traveling to target objects with minimal a priori knowledge of the environment. Given the sparse placement of goals in unknown scenes, exploration is essential for reaching remote objects of interest that are not immediately visible to autonomous agents. Sparse rewards are a crucial problem in reinforcement learning that arises in object goal navigation, as positive rewards are only attained when targets are found at the end of an agent's trajectory. As such, this work explores object goal navigation and the challenges it presents, along with the relevant reinforcement learning techniques applied to the task. An ablation study of the baseline approach for the RoboTHOR 2021 object goal navigation challenge is presented and used to guide the development of an on-policy agent that is computationally less expensive and obtains greater success in unseen environments. Then, original object goal navigation reward schemes that aggregate episodic and long-term novelty bonuses are proposed, and obtain success rates comparable to the respective object goal navigation benchmark at a fraction of training interactions with the environment.

Acknowledgements

I would like to thank my academic supervisor, Dr. Pierre Payeur, for supervising my research and providing me the space, opportunity, and guidance that inspired me throughout my research.

To my parents and partner, Emilie, I am truly grateful for your everlasting support, love, and motivation that encouraged me in this endeavour.

Table of Contents

List of Tables

List of Figures

Abbreviations

ALE arcade learning environment

CLIP contrastive language-image pretraining

CNN convolutional neural network

DD-PPO decentralized distributed proximal policy optimization

DQN deep Q-network

FMM fast marching method

GAE generalized advantage estimation

GPU graphical processing unit

GRU gated recurrent unit

HM3D Habitat-Matterport3D

ICM intrinsic curiosity module

IL-HD imitation learning with human demonstrations

KL Kullback-Leibler

KNN K-nearest neighbors

LSTM long short-term memory

MC Monte Carlo

MDP Markov decision process

MLP multilayer perceptron

MP3D Matterport3D

MSE mean squared error

NGU never give up

ObjectNav object goal navigation

OVRL offline visual representation learning

PB2 population-based bandits

PBL prediction of bootstrapped latents

PBT population-based training

POMDP partially observable MDP

PPO proximal policy optimization

R-CNN region-based CNN

R2D2 recurrent replay distributed deep Q-network

ReLU rectified linear unit

RGB red-green-blue

RGB-D red-green-blue-depth

RL reinforcement learning

RL-FT reinforcement learning fine tuning

RND random network distillation

RNN recurrent neural network

SGD stochastic gradient descent

SGE semantic goal exists

SPL success weighted by path length

TD temporal difference

THDA treasure hunt data augmentation

VAE variational auto encoder

Chapter 1: Introduction

1.1 Motivation

Advances in machine learning have made extraordinary contributions to a plethora of problems, such as natural language processing [1], autonomous navigation [2], and robotic motion [3]. Of particular interest are the developments of reinforcement learning (RL) in the field of object goal navigation.

Object goal navigation (ObjectNav) is the task of autonomously navigating to target objects in environments where a priori information of the environment is limited. As such, exploration plays a key role in being able to reach objects of interest that have unknown locations and are not immediately present in an autonomous agent's field of view.

It is natural that human-inspired solutions are explored to tackle autonomous navigation problems as humans excel in such tasks. Deep reinforcement learning, a field of research with conceptual origins in biological and behavioral sciences, has been successfully applied to autonomous navigation tasks [2] and can outperform classical methods. However, current state of the art in object goal navigation obtains a test success rate of roughly 65%[1] and 68%[2] on Habitat and RoboTHOR challenge environments [4, 5]. This is likely due to the sparsity of success rewards present in such problems. As such, object goal navigation remains a problem to be solved in a reinforcement learning framework.

RL approaches involve training agents to act given a state (i.e., RGB-D or pose observations) and a set of possible actions. Agents are trained to find an optimal policy, π^*,

[1]Retrieved from https://eval.ai/web/challenges/challenge-page/1615/leaderboard/3899.
[2]Retrieved from https://leaderboard.allenai.org/robothor_objectnav/submissions/public.

that maximizes the total expected reward over multiple interactions with an environment. When the set of possible states becomes overly complex, neural networks have been used to simplify state representations [6], estimate state values, and learn policies [7,8]. Over recent years, advances in deep reinforcement learning have allowed researchers to train agents to outperform humans in tasks such as video games [9, 10], Go, shogi, and chess [11].

However, sparse rewards are a fundamental problem in RL that arise in object goal navigation since positive rewards are only attained when target objects are found at the end of an agent's trajectory. Sparse rewards make it difficult for an agent to assign value to past states and actions based on rewards far into the future. Reward engineering and environment modelling is one approach to solve this problem; although, the process is time consuming and can unintentionally bias agent action distributions to undesirable behaviors. Furthermore, the dynamics of the environment are often unknown; hence, the optimal reward configuration of the environment remains a substantial problem in itself.

Humans, in contrast, are exceptional at completing tasks even when rewards are non-existent or present themselves far into the future. Intrinsic motivations play a sizable role in guiding our actions towards goals that are inherently satisfying, as opposed to externally consequential. Curiosity is an intrinsic motivator that enhances human learning [12]; hence, efforts endowing agents with curiosity have been explored in the field of reinforcement learning. Some successful methods that apply intrinsic curiosity to sparse reward RL environments provide a bonus to the agent based on some formulation of novelty or surprise. Some of the top performers in common sparse reward RL tasks such as Montezuma's Revenge [9] or VizDoom [13] represent intrinsic bonus rewards as pseudo-counts [14, 15], or state prediction errors [16, 17], or a combination of both [18, 19]. These approaches can be

implemented with a variety of RL algorithms and allow for end-to-end training of agents that are effective at exploring environments even in the absence of extrinsic (environmental) rewards.

Count-based methods, which approximate an intrinsic reward by the inverse of a state visitation count, have been applied in object goal navigation [20]. Similarly, Maksymets et al. [21] use the increase in area observed by an agent as an exploration bonus. However, these approaches still leave room for improvement in their respective challenge benchmarks and rely on effective measurement and mapping data, which are not always available in unknown environments.

When used in conjunction with extrinsic rewards, prediction error bonuses can influence agents to effectively explore a given environment while also learning to complete the task at hand. Intrinsic curiosity module (ICM) [17] and random network distillation (RND) [16] methods estimate state prediction errors using neural networks. While ICM predicts future states using a forward model, RND estimates the output of a fixed target network using a predictor network. Despite their success in sparse reward settings, intrinsic curiosity prediction error bonuses are yet to be fully explored in the context of object goal navigation.

1.2 Problem Statement

The aim of this research is to investigate the use of intrinsic curiosity via prediction error and count-based methods for object goal navigation. This research proposes the use of curiosity as an appropriate exploration method that can improve performance in finding

targets of interest that are not immediately present in a mobile robot's field of view. Moreover, intrinsic rewards could help a robotic agent learn an effective exploration strategy to discover the sparse rewards existent in the ObjectNav task. The scope of this research is limited to training agents to generalize in unseen simulated environments during object navigation tasks assuming the availability of RGB and depth sensors, and does not attempt to address the discrepancies associated when porting agents to real world environments, a problem known as the sim-2-real gap. The research is also limited by resources when compared to common reinforcement learning benchmarks in object goal navigation. Typical benchmarks experiment for over 300M+ environment interactions, over distributed systems with several GPUs and capacity to run multitudes of environments in parallel. Hence, this research tries to show performance improvements in the short run (10-50M environment steps), as opposed to running experiments on expensive systems for extended periods of time.

1.3 Objectives

The aim of this research is to:

- Implement a robotic agent to navigate to objects outside its immediate sensing range in a benchmarked 3D simulated navigation environment.
- Explore and design methods of intrinsic curiosity in autonomous mobile agents for the task of object goal navigation.

- Experimentally validate the use of exploration bonuses with state-of-the-art reinforcement learning agents on object goal navigation tasks.

1.4 Thesis Overview

Chapter 2 reviews the relevant literature of artificial neural networks and deep reinforcement learning from the perspective of object goal navigation. Chapter 3 details the proposed architecture and experimental setup of on-policy agents in the RoboTHOR [5] 3D simulated environment. Chapter 4 details curious RL agents and compares them against non-curious ones, while Chapter 5 concludes on this thesis and its findings.

Chapter 2: Literature Overview

This chapter overviews relevant literature regarding deep reinforcement learning and object goal navigation. The relevant fundamentals of artificial neural networks are reviewed in section 2.1 and reinforcement learning along with the landscape of deep RL algorithms are explored in section 2.2. Section 2.3 looks at the current literature regarding object goal navigation with a focus on reinforcement learning approaches.

2.1 Artificial Neural Networks

The objective of machine learning is to identify a model that best predicts an output $Y \in \Re^N$ given the features $X \in \Re^M$ by learning on a dataset $D = \{(x_1, y_1), \dots (x_n, y_n)\}$. Neural networks are a class of models optimized to predict a function $Y = F(X)$, most commonly optimized via stochastic gradient descent, or some version of it. A loss function, such as the mean squared error (MSE), defines the metric by which to optimize network weights. The following sub-sections overview different neural network architectures and training techniques used to best fit D and generalize to $F(X)$.

2.1.1 Architectures

A single neuron is defined as a weighted linear combination of input features activated by some, often non-linear, activation function (e.g. sigmoid, hyperbolic tangent). Common network architectures build on this basic formulation by staging sequential layers to produce large, highly parametrized models capable of predicting complex non-linear functions.

Multilayer Perceptron

The multilayer perceptron (MLP) [22] structures a network of neurons that feed stimuli from a previous layer onto the next sequentially. The dimension of each hidden layer is commonly referred to as the *width* of the layer, and the number of layers in the network is referred to as the *depth*. The output of each layer is defined as

$$h^{(l)} = g(W^{(l)}h^{(l-1)} + b^{(l)}) \tag{2.1}$$

where $h^{(l)}$ is the output vector of hidden layer l, g is the activation function, $W^{(l)}$ is a 2D matrix of learnt parameters with dimensions defined by the previous and current layer widths, and $b^{(l)}$ is a learnt bias term.

Convolutional Neural Networks

The main building block of a convolutional neural network (CNN) [23] is a discrete convolutional layer with a kernel $W \in \Re^{k_1 \times k_2 \times C}$ where k_1, k_2 are the kernel width and height respectively, and C is the number of channels of the input features (e.g. 3 for an RGB image in the input layer).

Many CNN architectures have been proposed for image classification. AlexNet [24] uses rectified linear unit (ReLU) activation functions, dropout and pooling layers. ReLU activations output the input when positive and zero otherwise, dropout is a regularization technique which randomly zeros neuron outputs when training, and pooling layers reduce

feature dimensionality with a pooling operation (max, mean, min, etc.). VGG [25] introduces deeper homogeneous networks with smaller 3x3 kernels leading to performance gains over AlexNet in the ImageNet classification problem. ResNet [26] introduces identity residual connections between input features and the output of convolutional layers. The CNN defined by Lecun et al. [23] and the ResNet architecture defined by He et al. [26], which is often used as a backbone architecture in many computer vision models, are further described in appendix A.1.

CNNs also enable object detection, which requires identifying objects in the frame and where in the frame they are located. The region-based CNN (R-CNN) [27] proposes a set of objects in the scene via selective search, and then feeds the resized region to a CNN pretrained on classification. Faster R-CNN [28], the first end-to-end object detection network, improved speeds with a region proposal network to identify potential objects while also performing bounding box regression. Mask R-CNN [29] further improves on this by outputting a pixel-wise binary map indicating the existence of an object, extending the Faster R-CNN architecture to object segmentation.

In many computer vision problems, it can be useful to repurpose models trained for a particular problem, such as ImageNet [30] classification, and apply them to different learning problems, known as transfer learning. Contrastive language-image pretraining (CLIP) is a pretraining method which learns embeddings of image and language models via contrastive objectives [31]. Backbones used are the Vision Transformer [32] and ResNet architectures, which are trained on a large database of 400 million image-text pairs from the internet. The resulting CLIP models achieve zero-shot prediction performance on several classification datasets that surpasses some models trained strictly on those datasets. CNN

transfer learning is currently used in state-of-the-art object goal navigation approaches [33, 34].

Recurrent Neural Networks

Recurrent neural networks (RNNs) are commonly used to model sequential data as they maintain a relationship between sequential activations. An RNN consists of recurrent units with a hidden state h_t representing the unit's state at step t. Elman [35] introduces vanilla RNN governed by equations (2.2) and (2.3).

$$h_t = g(Wx_t + Uh_{t-1} + b) \tag{2.2}$$

$$y_t = Vh_t + c \tag{2.3}$$

where $W \in \Re^{k \times m}, U \in \Re^{k \times k}, b \in \Re^k, V \in \Re^{n \times k}, c \in \Re^n$, and n, k, and m are the dimensions of the output y_t, the state h_t, and the input x_t, respectively.

RNNs are known to suffer from vanishing or exploding gradients when backpropagating errors. Some architectures consist of more complex recurrent units and additional inter-unit signals that aim to reduce the effects of vanishing or exploding gradients. Long short-term memory (LSTM) [36] and gated recurrent unit (GRU) [37] networks are examples of such architectures which consist of gating mechanisms per recurrent unit. Further details of their architectures can be found in appendix A.2.

RNNs are commonly used in reinforcement learning to model the temporal relationship between sequential environment interactions and are used in several object goal navigation

approaches [20, 33, 34, 38].

2.1.2 Optimization

A loss or score function defines the metric to optimize network weights and an optimization technique optimizes them. Stochastic gradient descent (SGD), or some version of it, is a commonly chosen optimization method for neural networks, although, other methods such as evolutionary strategies can also optimize neural networks effectively [39]. Common loss functions include mean squared error (MSE), cross-entropy, and Kullback-Leibler (KL) divergence [40] losses. Cross entropy is commonly used in classification problems, MSE is commonly used in regression, and KL divergence is used to measure the distance between predicted and target distributions.

Network weights are optimized via backpropagation [41] with a gradient descent method by computing loss gradients with respect to weights. Mini-batched stochastic gradient descent optimizes networks based on equation (2.4). SGD with momentum accelerates learning by reducing oscillations [42]. AdaGrad decays learning rates based on the sum of squares of the gradient [43]. Adam similarly computes decaying learning rates using averages of past gradients and their square [44].

$$\theta_{new} = \theta_{old} - \eta \frac{1}{|B|} \sum_{B} \boldsymbol{\nabla}_{\theta} \mathcal{L}_{\theta}(Y, \hat{Y}) \tag{2.4}$$

θ denotes the network weights, η is the learning rate, $\mathcal{L}_{\theta}$ is the loss function and Y and $\hat{Y}$ are the batched targets and network outputs respectively.

2.2 Reinforcement Learning

Reinforcement learning (RL), in the context of machine learning, is the training of models to make decisions using observations of an environment state by reinforcing or penalizing actions with a reward signal. This section reviews the basic formulation of the Markov decision process (MDP), model-free reinforcement learning, deep reinforcement learning, and curiosity bonus rewards in RL.

2.2.1 Markov Decision Processes

Although not all RL problems consist of Markov states, the MDP formulation is the starting point for many reinforcement learning algorithms. MDPs are defined by a set of Markov states S, a set of actions A, the state transition probability, $p(s'|s,a)$, a reward function $r(s,a,s')$, and the discount factor γ, where $s, s' \in S$ are current and next states, and $a \in A$ is the action transitioning the agent from s to s'. An agent acting in a MDP sequentially interacts with the environment and observes the next state and reward. The goal of the agent is to maximize rewards by optimizing a policy π that maps agent observations to actions given current state observations. Value functions estimate the value of states or state-action pairs of a MDP, which is often used in RL to learn optimal policies.

The State

The set of states S is considered Markov if future states in a trajectory are independent of the past states. That is,

$$Pr\{R_{t+1} = r, S_{t+1} = s' | S_t, A_t\} = Pr\{R_{t+1} = r, S_{t+1} = s' | S_0 A_0, R_1 ... S_t A_t, R_t\} \quad (2.5)$$

where R_t, S_t, A_t are the reward, state and action at step t, respectively [45]. Hence the Markov state encapsulates sufficient information about previous states to act optimally in the environment. Often in RL, state observations are non-Markov (i.e., future states are dependent on past states). Nonetheless, it is still possible to learn policies that complete tasks despite such observations. Finite MDPs have state and action spaces that are finite, which is the case of object goal navigation with discrete actions and discrete red-green-blue (RGB) data. Partially observable MDPs (POMDPs) only allow for partial state signals, such as the subsection of a maze in an agent's locale. Still, approximations of a state signal can lead to successful policies. In the context of autonomous navigation, environments are partially observable and states are non-Markov, however approximations of the environment state can be sufficient to solve tasks such as point goal navigation [2].

State-Reward Transition Models

State-reward transition probabilities model the future transitions with the environment and represent the dynamics of a MDP. Knowledge of a MDP's dynamics allows for solving the task directly, although more often than not, the environment dynamics are unknown.

Following [45], the state-reward and state transition models, along with expected reward functions are:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{2.6}$$

$$p(s'|s, a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_r p(s', r|s, a) \tag{2.7}$$

$$r(s, a, s') = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \sum_r r\frac{p(s', r|s, a)}{p(s'|s, a)} \tag{2.8}$$

$$r(s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] = \sum_r r \sum_{s'} p(s', r|s, a) \tag{2.9}$$

The state-reward transition model (2.6) denotes the probability of achieving reward r and observing state s' when transitioning from state s after action a. The state transition model (2.7) is the probability of observing state s' when transitioning from state s after action a. Equation (2.8) denotes the expected reward given s, a, and s', and equation (2.9) denotes the expected reward given state s, and action a.

In non-complex MDPs these models are often well defined; however, in a large portion of RL problems, including POMDPs, they are not. Model-based reinforcement learning algorithms concern themselves with modeling the environment state transition models (e.g., world models [6]). Model-free reinforcement learning algorithms do not assume the dynamics of the environment are known.

Agent Policies

The agent's policy maps state observations to a distribution of actions in the set A. These policies can be deterministic ($a = \pi(s)$) or stochastic ($a \sim \pi(a|s)$). Agent actions are

sampled from their policies and can be either continuous, like the angle of a motor, or discrete, like "rotate right" or "move ahead" in the context of discrete navigation. Policies are often parametrized functions, denoted π_θ, especially in policy gradient methods (e.g., A3C [7], TRPO [46], PPO [8], etc.).

Value Functions

Value functions represent the expected total future rewards given a current state or state-action pair. It is often desirable to represent the value of a state or state-action pair by discounting the cumulative rewards. Hence the return, G_t, is defined as the discounted reward over future interactions of an agent's trajectory. The factor $\gamma \in [0, 1]$ discounts future rewards, modulating the influence of future rewards on past state values.

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.10}$$

Hence the state value, $v_\pi(s)$, and the state-action value, $q_\pi(s, a)$ are defined as [45]:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{2.11}$$

$$= \sum_a \pi(a|s) q_\pi(s, a) \tag{2.12}$$

$$= \sum_a \pi(a|s)(r(s, a) + \gamma \sum_{s'} p(s'|s, a) v_\pi(s')) \tag{2.13}$$

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s, A = a] = \mathbb{E}[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A = a] \tag{2.14}$$

$$= r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s') \tag{2.15}$$

$$= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_{a'} \pi(a'|s') q_\pi(s',a') \tag{2.16}$$

These are known as the Bellman equations. Optimal value functions of a MDP are obtained when an agent is acting under an optimal policy π^*. That is, an optimal policy is one that maximizes the value of all states; hence, the MDP is solved when the optimal value function is known. The Bellman optimality equations are starting points for many reinforcement learning algorithms as many solutions compute optimal state value functions to obtain an optimal policy [45]. The optimal value functions are defined as:

$$v^*(s) = \max_\pi v_\pi(s) \tag{2.17}$$

$$= \max_a q^*(s,a) \tag{2.18}$$

$$= \max_a \left[r(s,a) + \gamma \sum_{s'} p(s'|s,a) v^*(s') \right] \tag{2.19}$$

$$q^*(s,a) = \max_\pi q_\pi(s,a) \tag{2.20}$$

$$= r(s,a) + \gamma \sum_{s} p(s'|s,a) v^*(s') \tag{2.21}$$

$$= r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} q^*(s',a') \tag{2.22}$$

2.2.2 Model-Free Reinforcement Learning

Model-free reinforcement learning assumes the dynamics of the environment are unknown; hence, the expected future returns are estimated empirically via interactions with the environment. Monte Carlo (MC) and temporal difference (TD) methods approximate state and state-action value functions while updating an online or offline policy.

Monte Carlo and Temporal Difference

Monte Carlo methods estimate state-action values as the average cumulative returns over trajectories [45]. The value function update is a running mean that counts state visitations, although a learning step coefficient α can be used instead of dividing by the number of samples. Equation (2.24) denotes the Monte Carlo (MC) state-action value function update, which is parametrized with a step size, α.

$$Q(s, a) = \mathbb{E}\left[G_t | S_t = s, A_t = a\right] \tag{2.23}$$

$$Q(s, a) = Q(s, a) + \alpha(G_t - Q(s, a)) \tag{2.24}$$

Temporal difference (TD) approaches estimate state-action values as the average *estimated* cumulative reward [45]. Instead of a target of G_t, n-step TD estimates approximate the future return looking n steps into the future. Equations (2.25) and (2.26) show the 1-step

and n-step TD state-action value function update respectively.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.25}$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(G_t^{(n)} - Q(s_t, a_t)) \tag{2.26}$$

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \ldots \gamma^n Q(s_{t+n}, a_{t+n}) \tag{2.27}$$

TD(λ) methods make use of multiple n-step returns ($G_t^{(n)}$) by modulating them with decreasing weights. Equation (2.29) describes the TD(λ) state-action value estimate update. When $\lambda = 1$, the state-action error amounts to Monte Carlo error [45].

$$G_t^{(\lambda)} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{2.28}$$

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(G_t^{(\lambda)} - Q(s_t, a_t)) \tag{2.29}$$

SARSA [47] is an on-policy algorithm that makes use of temporal difference state-action value estimates. At each step, $Q(s, a)$ is updated following an ϵ-greedy policy that acts randomly with uniform probability ϵ or acts greedily otherwise. Q-learning is an off-policy algorithm with temporal difference estimates that optimizes state-action values assuming a greedy policy in future actions but following an ϵ-greedy policy [48]. Refer to algorithms 2.1 and 2.2 in appendix B.1 for details on SARSA and Q-learning.

2.2.3 Deep Reinforcement Learning

In the case that the state space of a MDP is dimensionally large, tabular state-action value functions become prohibitively expensive to compute. Hence, value functions can

be estimated via function approximations such as linear regression models, or in the case of deep reinforcement learning, neural networks. In addition to a parametrized value function, policies can also be functions that map actions stochastically or deterministically from observations of the environment state.

Approximated value functions, $V(s, \theta)$ or $Q(s, a, \theta)$, learn the estimated discounted return, where θ are the function parameters to be learnt. A common loss function is the mean squared error between the learnt value function and the actual state-value. However, since the actual value function is not known, MC or TD estimates are appropriately used to estimate the future discounted return. The loss and its gradient can be defined as

$$\mathcal{L} = \mathbb{E}[(G_t - V(s, \theta))^2 | s] \tag{2.30}$$

$$\boldsymbol{\nabla}_\theta \mathcal{L} = \mathbb{E}[2(G_t - V(s, \theta)) \boldsymbol{\nabla}_\theta V(s, \theta) | s] \tag{2.31}$$

where G_t can be estimated by any of MC or TD methods described in section 2.2.2. The value function network is optimized with backpropagation via a gradient descent optimization method using this loss definition.

Deep Q-learning

The vanilla deep Q-network (DQN) optimizes $Q(s, a, \theta)$ by interacting with an ϵ-greedy policy while optimizing $Q(s, a, \theta)$ assuming a greedy policy [49]. In addition, an experience replay buffer holds environment interactions for re-use of previous experiences and the future discounted rewards are estimated with TD(0) estimates. The loss function and its

gradient are denoted

$$\mathcal{L}(\theta_i) = \mathbb{E}[(r_t + \gamma \operatorname*{argmax}_{a'} Q(s', a', \theta_{i-1}) - Q(s, a, \theta_i))^2 | s, a] \tag{2.32}$$

$$\boldsymbol{\nabla}_{\theta_i} \mathcal{L}(\theta_i) \approx \mathbb{E}[(r_t + \gamma \operatorname*{argmax}_{a'} Q(s', a', \theta_{i-1}) - Q(s, a, \theta_i)) \boldsymbol{\nabla}_{\theta_i} Q(s, a, \theta_i) | s, a] \tag{2.33}$$

where θ_i denotes the weights of training iteration i. Double DQN [50] adapts double Q-learning from the tabular setting to DQN. Dueling DQN [51] learns a Q-function as $Q(s, a, \theta, \alpha, \beta) = V(s, \theta, \alpha) + A(s, a, \theta, \beta)$ where θ are the shared parameters between V and A, α and β are the parameters unique to V and A respectively, and A is the advantage function.

Policy Gradients

Policy gradient reinforcement learning algorithms approximate policy $\pi(a|s, \theta)$ as a differentiable function parametrized by θ. One objective function is to maximize the expected un-discounted return of trajectories. Hence

$$\mathcal{J}(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} R_{t+1} \right] = \sum_{\tau} p(\tau|\theta) \sum_{t=0}^{T} R_{t+1} \tag{2.34}$$

where τ denotes a trajectory. The gradient of the objective for a batch of trajectories

is

$$\nabla_\theta \mathcal{J}(\theta) = \nabla_\theta \mathbb{E}_\pi \left[\sum_{t=0}^{T} R_{t+1} \right] \tag{2.35}$$

$$= \sum_\tau \nabla_\theta p(\tau|\theta) \sum_{t=0}^{T} R_{t+1} \tag{2.36}$$

$$= \sum_\tau p(\tau|\theta) \nabla_\theta \ln p(\tau|\theta) \sum_{t=0}^{T} R_{t+1} \tag{2.37}$$

$$= \mathbb{E}_\pi \left[\nabla_\theta \ln p(\tau|\theta) \sum_{t=0}^{T} R_{t+1} \right] \tag{2.38}$$

$$= \mathbb{E}_\pi \left[\sum_{t'=0}^{T} \nabla_\theta \ln \pi(a_{t'}|s_{t'}, \theta) \sum_{t=0}^{T} R_{t+1} \right] \tag{2.39}$$

where

$$p(\tau|\theta) = p(s_0) \prod_{t=0}^{T} p(s_{t+1}|s_t, a_t) \pi(a_t|s_t, \theta) \tag{2.40}$$

$$\ln p(\tau|\theta) = \ln p(s_0) + \sum_{t=0}^{T} \ln p(s_{t+1}|s_t, a_t) + \sum_{t=0}^{T} \ln \pi(a_t|s_t, \theta) \tag{2.41}$$

$$\nabla_\theta \ln p(\tau|\theta) = \sum_{t=0}^{T} \ln \nabla_\theta \pi(a_t|s_t, \theta) \tag{2.42}$$

Equation (2.39) is derived assuming un-discounted rewards for a full trajectory; however, a similar formulation can be made considering future discounted rewards as the target. The target, denoted Φ_t, can also be chosen to reduce the variance of the expectation by using value functions, advantages or TD residuals. As examples, generalized advantage estimation uses a method analogous to TD(λ) to estimate advantages as targets [52], while

the Monte Carlo REINFORCE algorithm [53] estimates value function targets. Equation (2.43) re-formulates equation (2.39) generally with target Φ_t.

$$\nabla_\theta \mathcal{J}(\theta) = \mathbb{E}_\pi \left[\sum_{t'=0}^{T} \nabla_\theta \ln \pi(a_{t'}|s_{t'}, \theta) \Phi_t \right] \tag{2.43}$$

Some actor-critic methods use parametrized value approximations for computations of Φ_t. Asynchronous advantage actor-critic (A3C) [7] estimates the advantage function

$$\Phi_t = A(s_t, a_t, \theta_v) \tag{2.44}$$

$$= G_t - V(s_t, \theta_v) \tag{2.45}$$

where θ_v denotes the state value function parameters. Each thread of A3C is synchronized with global policy and value function parameters, and episodes of agent interactions with the environment are executed in parallel. Gradients are accumulated per episode and are used to update the global parameters. A3C also uses the policy entropy in the loss objective to encourage agent exploration.

Trust region policy optimization [46] similarly estimates advantages but also bounds policy updates by computing the KL-divergence between updated and previous policies. The KL-divergence is constrained to some target value δ for the surrogate loss function:

$$\mathcal{J}(\theta) = \mathbb{E}_\pi \left[\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})} A_t \right] \tag{2.46}$$

Proximal policy optimization (PPO) [8] similarly bounds policy updates by optimizing the clipped surrogate loss function

$$\mathcal{J}(\theta) = \min\left(\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})} A_t, clip\left(\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})}, 1-\epsilon, 1+\epsilon\right) A_t\right) \tag{2.47}$$

where ϵ clips the ratio between the new and old policies.

2.2.4 Curiosity Bonus Rewards

Curiosity in the domain of reinforcement learning attempts to solve the problem of sparse rewards by reinforcing agent policies to explore the environment effectively. This is achieved by supplementing reward signals with bonuses that encourage exploration.

Count-Based Bonuses

Count-based methods characterize novelty as inversely proportional to some count denoted N_t. In tabular MDPs, counts are easily recorded and the reward bonus is characterized as

$$r_t^i(s, a) \propto \frac{\beta}{\sqrt{N_t(s, a)}} \tag{2.48}$$

$$r_t^i(s) \propto \frac{\beta}{\sqrt{N_t(s)}} \tag{2.49}$$

where β is an intrinsic reward coefficient.

It follows that the bonus reward is largest when state or state-action visitation counts are lower (i.e. novel). Sutton [54] proposes DynaQ+, which uses Q-learning with a count

based novelty reward. However, in MDPs with large or continuous state and action spaces, it is not viable to accumulate counts in this fashion. In that case, function approximations can be used to estimate the visitation counts. In [14], Bellemare et al. propose pseudo-counts computed via equation (2.50). The prediction gain is computed as the difference in log-probabilities of current and next step state densities, $\rho(s)$ and $\rho'(s)$, which are parametrised by some function (Context Tree Switching [55] in the original paper). The pseudo-count is then approximated as

$$\hat{N}_t(s) = \left(e^{\log \rho'(s) - \log \rho(s)} - 1\right)^{-1} \tag{2.50}$$

Ostrovski et al. [56] build on the pseudo-count approach by approximating state densities with a modified PixelCNN [57] and clipping the prediction gain to positive values.

In another count-based approach, Tang et al. [58] use an autoencoder to learn a latent binary representation of the state which can then be counted in a tabular fashion. The intrinsic reward is represented as

$$r^i(s) = \frac{\beta}{\sqrt{N(\phi(s))}} \tag{2.51}$$

$$\phi(s) = \text{sgn}(Ag(s)) \in \{-1, 1\}^k \tag{2.52}$$

where $g : S \to \mathfrak{R}^D$ is a preprocessing function, ϕ is the hash code for tabular counts, A is a $k \times D$ matrix with values sampled from a normal distribution with zero mean and unitary standard deviation, and k is the hash code's dimension which controls the granularity.

Prediction Error Bonuses

Bonus rewards can also be characterized as prediction errors of states. As the agent interacts with the environment, the prediction error is expected to decrease for states that have already been visited. In [17], Pathak et al. estimate the state prediction error with an intrinsic curiosity module (ICM) that learns a forward model of states while also learning the inverse dynamics of an environment.

The difference between the output of the forward model and the actual observed state is added to the reward signal. The cross entropy between the inverse model's predicted action and the actual action that was taken is the learning signal for the feature encoder $\phi(s)$. Burda et al. [59] further investigate ICM rewards using different feature embeddings, including inverse dynamics, variational auto encoder (VAE) and random features, across 54 games. They show that an agent trained solely on ICM rewards can attain extrinsic rewards and that inverse dynamics, VAE, or random features are useful for learning in many of the environments tested on.

Similarly, random network distillation (RND) supplements the reward signal with a prediction error computed as the mean squared error between the outputs of target and prediction networks [16]. The predictor is trained to estimate the output of a fixed target network, hence the intrinsic reward decreases as the predictor network better mimics the target.

Badia et al. [18, 19] use both count and prediction type bonuses to model episodic novelty and long term curiosity. An episodic count is computed with a kernal function implemented via K-nearest neighbors (KNN) and an episode memory buffer. RND is used

to model long term curiosity and modulate episodic intrinsic rewards. The intrinsic reward function is modeled by:

$$r_t^i = r_t^{episodic} \cdot \min(\max(\alpha_t, 1), L) \tag{2.53}$$

$$\alpha = 1 + \frac{err(\Phi(s_t), \hat{\Phi}(s_t)) - \mu_{err}}{\sigma_{err}} \tag{2.54}$$

$$err(\Phi(s_t), \hat{\Phi}(s_t)) = \frac{\sum(\Phi(s_t) - \hat{\Phi}(s_t))^2}{n} \tag{2.55}$$

$$r_t^{episodic} = \frac{1}{\sqrt{\sum_{s_i \in M} K(s, s_i)} + c} \tag{2.56}$$

where α modulates the episodic intrinsic reward, $r_t^{episodic}$, L is an upper bound to α, M represents the episodic memory, Φ and $\hat{\Phi}$ are the RND target and predictor networks, err is the RND mean squared error, n denotes the flattened dimension of the RND network outputs, c is some small constant, and K is the kernel function implemented as k-nearest neighbors. Badia et al. [18] highlights that RND modulation of episodic rewards have limited performance improvement on dense reward settings while also demonstrating pure exploration agents capable of superhuman average performance on a number of Atari games, which confirms the findings in [59]. Badia et al. [18, 19] detail off-policy agents that train multiple value networks and policies that sample exploration coefficients, β_i; hence, the experience replay buffer contains trajectories varying in exploratory behaviours.

2.3 Object Goal Navigation

ObjectNav is the task of navigating to a target object in an unknown environment without a priori information starting from a random location. A robotic agent, equipped with useful sensory input such as red-green-blue-depth (RGB-D) and pose, must intentionally stop in front of the target object to be successful, and ideally do so in an efficient manner. Success is achieved if the agent stops within 1m of the target while maintaining target visibility.

2.3.1 Challenge Environments

Multiple simulated environments have been developed, each with differing implementations with respect to scenes, target objects, and embodied agents.

Simulated environments for object goal navigation aim to model the real world's complexity and semantics to mitigate the challenge of porting a learned policy to a physical agent. As such, multiple datasets are created from 3D scans of physical environments, which enables generalization testing on agents transferred to their real world counterparts.

Habitat

The Habitat ObjectNav challenges are based on the Matterport3D (MP3D) [60] dataset which includes 90 indoor scenes of real world homes and buildings that have been scanned with 40 annotated categories of segmented objects and scene components. The newest 2022

iteration of the habitat challenge uses the Habitat-Matterport3D (HM3D) [61] dataset, which has an 80/20/20 train/val/test split and uses 6 target objects (chair, couch, potted plant, bed, toilet and TV). This is a change from previous iterations where a set of 21 object categories were used, likely facilitating the latest Habitat challenge.

Agent characteristics in this challenge are based on the LoCoBot[1] and sensing capabilities comprise of noiseless RGB-D, GPS and compass data. The action space is discrete: *move-forward 0.25m, turn-left 30°, turn-right 30°, look-up 30°, look-down 30°* and *stop*, and success is achieved when the agent stops within 1m of the target object while occupying a valid viewpoint. A valid viewpoint of the object is considered any point within 1m of the target object that *could* allow for framing of the object, although object visibility is not required. Figure 2.1 shows sample floor layouts, sensory inputs, and object class segmentation of the MP3D dataset.

[1]For details on the LoCoBot, see http://www.locobot.org/.

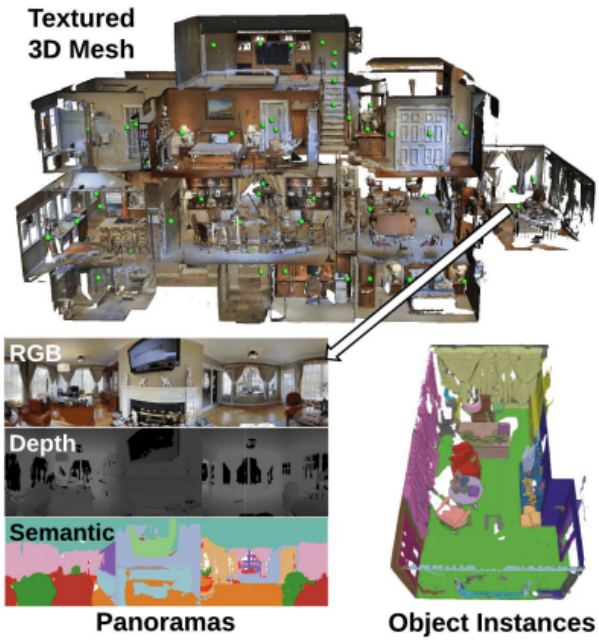

Figure 2.1: Sample from the MP3D dataset.

RoboTHOR

The RoboTHOR environment is a Unity [62] powered simulation for embodied AI development that contains 89 apartment layout scenes (60 train, 15 val, 4 test-dev, and 10 test-standard) [5]. There are 12 target object categories that can be selected for each scene, and several non-target objects in the scene (a total of 43 object categories).

The RoboTHOR agent is also based on the LoCoBot and has similar sensing as in Habitat challenges, although pose information is unavailable. The agent has the same action space as in Habitat, and has 500 actionable steps to stop within one meter of the target with the object in view. Dissimilar to Habitat, the RoboTHOR challenge only

considers a trajectory successful if the agent is within 1m of the target object *and* it maintains visibility of the object when executing the "Stop" action. The target categories are also different with little overlap: AlarmClock, Apple, HousePlant, Television, Basketball, Laptop, Bowl, GarbageCan, Mug, Vase, BaseballBat, SprayBottle. Compared to Habitat, the RoboTHOR target objects vary greatly in size. Another important difference to note is the different constructions of scenes (Unity vs MP3D); observations in MP3D visually resemble noisy real world images more so than Unity RGB-D frames. Figure 2.2 shows samples from the AI2-THOR framework, which the RoboTHOR challenge is built upon.

Figure 2.2: Sample from the AI2-THOR framework.

2.3.2 Metrics

Object goal navigation agents are commonly evaluated using success and success weighted by path length (SPL) rates on a test set of episodes. While the success rate measures the

agent's raw ability to complete the ObjectNav task, the SPL of an episode is commonly used for evaluating an agent's trajectory efficiency to successfully navigate to the target object. An episode's SPL is defined as

$$SPL = S \times \frac{l}{\max(p, l)} \tag{2.57}$$

where $S \in \{0, 1\}$ indicates the episodes success, l is the length of the shortest path, and p is the path length of the agent's trajectory. Although p should always be greater than or equal to l, the maximum of the two ensures that the SPL is normalized to have a maximum of 1, potentially accounting for structural biases in the environment that allow the agent to find shortcuts to the target object. Despite it being the most commonly used efficiency metric, SPL is limited in evaluating unsuccessful trajectories and is difficult to compare from one environment or dataset to another [63].

In the context of ObjectNav, training success rates and SPLs relate to the agent's performance on scenes it has already encountered, while validation and test metrics evaluate the agent on unknown scenes. The difference between training and test/validation sets is referred to as the generalization gap and is indicative of the agent's ability to generalize to unseen environments. Training metrics presented in this thesis relate to the agent's performance on a batch of training interactions, not the full training dataset. Hence, the agent's true success rate or SPL generalization errors, which would instead be evaluated using the full training dataset, are not presented. This is due to the costly time it would take to evaluate performance metrics on the full training dataset. Still, any large generalization gaps presented in this thesis do indicate some level of overfitting to the training dataset, while smaller errors could be simply attributed to the object class or episode difficulty

imbalance of the training batch.

ObjectNav agent performances are commonly compared by the checkpoint which obtains the greatest success rate or SPL. In this thesis, performance metrics are reported for the agent model checkpoint that obtains the greatest validation success rate. Additionally, the learning dynamics of the agent are shown for both success rate and SPL to demonstrate the agent's progression throughout training. If a plateau is reached, it could indicate training saturation. Training metric curves report the 64-window rolling average of the training batches of interactions with a 95% confidence interval as a method of demonstrating the general trend of what would otherwise be noisy training curves. Validation curves report the agent's performance on the full validation dataset and are presented for checkpoints taken during training without any averaging. In machine learning applications, configurations are normally run multiple times and their averages are reported to account for stochasticity between runs; however, due to time constraints and processing requirements of the simulation environment, single runs are reported. This is not uncommon as leaderboard benchmarks and ObjectNav approaches often report on their best performing agents.

While some navigation challenges, such as point goal navigation, have already been technically solved (at least in simulation), as done so by Wijmans et al. [2], test success rates on the ObjectNav task have only reached 65% and 68% on Habitat and RoboTHOR challenge environments, respectively. This is likely due to the increased complexity added by requiring the agent to intentionally stop with the target object in view, which necessitates some interpretation of semantics. During the writing of this thesis, strides have been made in other works to improve upon ObjectNav methods, often by addressing poor gen-

eralization on challenge datasets, as done so by Maksymets et al. [21], improving feature representation, as demonstrated by Khandelwal et al [34], or addressing reward sparsity, as detailed by Ramrakhya et al. [33]. Section 2.3.3 details some of the approaches to the ObjectNav task and their corresponding performance metrics.

2.3.3 Approaches

SemExp

The SemExp agent in [38] uses a semantic map with RGB-D sensors. The RGB frame is fed to a pretrained Mask R-CNN which explicitly outputs semantic labels while the depth image is converted to a point cloud. Each point in the cloud is associated with a label and depth and their composition is subsequently converted to a voxel representation. The sum of the voxels along the height forms a projection map containing a channel for each semantic category. The projection map is fed to a denoising network to produce a semantic map with the same dimensions. The Mask R-CNN and denoising network are trained using ground truth semantic segmentation and semantic maps respectively.

The semantic map is fed to a goal oriented semantic policy to determine a long term positional goal. The presence of the target object in the respective channel of the semantic map determines the output; if any non-zero cell indicating the presence of the object is found, they are set as the long term goal; otherwise, the semantic map is fed to a network that predicts where to explore. The goal network receives the semantic map, its positional trajectory, and the goal category. It is trained with PPO using the geodesic distance to

the nearest target object. A deterministic fast marching method [64] is used to plan a path to the long term goal.

The SemExp agent achieved 0.25 success rate and 0.10 SPL on the 2020 Habitat ObjectNav challenge, winning that year's competition.

Red-Rabbit

The RedRabbit agent aggregates RGB-D, semantic segmentation, goal descriptor, and pose sensor information to feed multiple RNN heads trained on different auxiliary tasks [20].

RedNet is used for segmentation [65] and a semantic goal exists (SGE) feature is calculated indicating the fraction of the target object occupying the agent's frame. Semantic channels are concatenated with RGB-D data and fed through a ResNet18 while the goal category is embedded into a 32 dimensional vector. The observation embeddings, composed of ResNet18 outputs, goal embeddings, pose, and SGE features, are processed by a set of recurrent belief modules implemented as GRU networks. Each GRU is associated with an auxiliary task and their outputs (the queries) are fused by applying an attention layer conditioned on the observation embeddings (the keys).

Ye et al. [20] train agent models on different tasks that do not directly optimize rewards, instead learning a task indirectly related to the agent's main objective. Six auxiliary tasks are used: two contrastive predictive coding [66], one prediction of bootstrapped latents (PBL) [67], one action distribution prediction, one generalized inverse dynamics, and one coverage prediction. In addition, rewards are augmented with a decaying episodic count-based novelty reward, computed by counting the agent's occupation in a voxel of the scene.

The rewards are integrated by a tethered policy; one policy head is optimized with the total combined reward (slack, success and exploration), and another is optimized on the sparse success reward. The agent is trained via PPO with importance weighted VTrace [68] using the full reward policy initially, then the sparse reward policy after a predetermined number of interactions. In addition to the PPO and auxiliary losses, both action and attention distribution entropies are added to the loss. The authors also note that a non-episodic intrinsic curiosity module (ICM) fails to change performance in their experiments.

The RedRabbit agent achieved first place in the 2021 Habitat ObjectNav challenge with 0.30 success rates and 0.13 SPL according to the challenge page; however the online leaderboard record shows a success rate of 0.237 and an SPL of 0.062.

Stubborn

Luo et al. [69] build on SemExp but make the following observations: 1) semantic segmentation does not aid in the SemExp agent's exploration, 2) there are many false positives of target detection based on a single frame, and 3) noisy sensors lead to inaccurate 2D maps leading to agents getting trapped. To deal with these issues, a simpler exploration goal selection strategy is followed, object detection is aggregated across frames, and 2D semantic maps are augmented with multi-scale collision maps. The exploration strategy is simplistic: move in one of 4 corner directions until a collision, after which the goal is set to another corner in a clock wise manner. Similar to the Red-Rabbit agent [20], RedNet [65] (as opposed to Mask R-CNN in SemExp) is used to create a 2D grid that estimates the existence of object categories. If the target is detected, the target's position is set as the

goal. Detection is verified with a binary Naive Bayes classifier pretrained on the MP3D dataset to reduce false positives. Aside from using different collision maps to avoid traps, the agent uses a brute force "rotate/move ahead" strategy when it is stuck in the same position for more than 10 steps.

The authors also make note that the collision avoidance approach used does not help the SemExp agent and hypothesize that the simplistic corner exploration strategy causes more frequent agent trappings. It still remains that using ground truth semantic segmentation, as demonstrated by Ye et al. [20], performs much better, indicating that noisy sensing is still an obstacle in semantic map based approaches.

The Stubborn agent achieves 0.237 success rate and 0.098 SPL, making the top 3 of the 2021 Habitat challenge leaderboard.

THDA

Treasure hunt data augmentation (THDA) [21] uses semantic segmentation using RedNet [65] along with depth and pose information to form its input features. The authors argue that the geodesic distance to the target object does not serve as an adequate dense reward for ObjectNav as it does for point goal navigation. It introduces an "ExploreTillSeen" reward scheme, which provides an exploration bonus, calculated as the percentage change in explored area of a 2D floor map, until the target object occupies more than 3% of the vision sensor's frame. They experiment with sparse success, dense geodesic, sparse with exploration, and "ExploreTillSeen" reward schemes. The ExploreTillSeen reward is described by equation (2.58)

$$r = r_{slack} + \begin{cases} r_{expl}\Delta_{\%explored} & \text{until goal seen} \\ r_{goal\_seen} & \text{first time goal is seen} \\ -\Delta_{geo} + Sr_{success} & \text{otherwise} \end{cases} \tag{2.58}$$

where Δ_{geo} is the change in geodesic distance between the agent and target object, $\Delta_{\%explored}$ is the percentage change in area explored, $S \in \{0, 1\}$ is the success flag, $r_{expl} = 25$, $r_{goal\_seen} = 3$, $r_{success} = 10$, and $r_{slack} = -0.01$. The change in exploration percentage is obtained by projecting the agent's view onto a 2D floor map of the environment and calculating the ratio of area observed at least once. It is worth noting that the exploration term can be considered an episodic intrinsic bonus, and its computation could be substituted for methods mentioned in section 2.2.4. This reward scheme greatly reduces the sparsity of positive rewards thanks to exploration bonuses as well as the "goal seen" term.

It is shown that dense geodesic rewards overfit the training episodes, the sparse reward agent fails to learn, the sparse with exploration scheme achieves similar results to the dense reward agent, and the ExploreTillSeen reward scheme somewhat reduces the train/test generalization gap. Additionally, the generalization gap is further reduced by limiting observations to depth and goal segmentation. The agent is trained on-policy with decentralized distributed proximal policy optimization (DD-PPO) without advantage normalization. Finally, the training set is augmented by inserting new objects into training scenes as a pre-training phase before fine-tuning on the challenge scenes. This further improves validation and test performances and shows that increasing the quantity of distinct training scenarios improves generalization.

THDA is fourth in the 2021 Habitat challenge leaderboard with an SPL of 0.089 and success rate of 0.214.

Habitat-Web

Ramrakhya et al. [33] use imitation learning from over 80 thousand human examples of completing the ObjectNav task, which incidentally provides a human benchmark for the Habitat 2021 challenge. Humans are able to achieve a 93% success rate and a 0.43 SPL when attempting the Habitat 2021 ObjectNav challenge. This highlights the issue of SPL being used as an efficiency metric for what's largely an exploration problem. The agent uses a PointNav pretrained ResNet50 [2] for depth observations, two ResNet18s for RGB observations and the semantic segmentation output of a pretrained RedNet [65], the agent's pose, and the semantic goal exists feature (as described in [20]).

The agent is trained using inflection weighted imitation learning [70] where trajectories are weighted by the inverse inflection frequency, calculated as the ratio of total actions in an "expert" trajectory, N, to the number of agent inflections (sequentially different actions), n_I. For all trajectories $T \in \{\tau_i\}_{i=1}^{N}$, the cross-entropy loss is characterized as

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \sum_{(s_t,a_t)\in\tau_i} -w_t \log(\pi_\theta(a_t|s_t)) \tag{2.59}$$

where the inverse inflection frequency, w_t, is

$$w_t = \begin{cases} N/n_I & a_t \neq a_{t-1} \\ 1 & \text{otherwise,} \end{cases} \tag{2.60}$$

a_t is the action, and s_t is the observation at timestep t.

The Habitat-Web agent obtains a success rates of 0.278/0.55 and SPLs of 0.099/0.22 on the 2021/2022 ObjectNav Habitat challenges respectively. The vast performance differences between the two challenges suggest a difficulty reduction in the 2022 Habitat challenge, likely due to the reduced number of target objects.

PONI

In [71], Ramakrishnan et al. use potential functions to estimate where to explore. Similar to the SemExp agent [38], the PONI agent is composed of a semantic mapping module, and uses a fast-marching policy [64] to reach long term goals. Dissimilar to the aforementioned approach, a potential function network, consisting of a U-Net [72] encoder, an area potential U-net decoder, and an object potential U-net decoder, is trained offline on semantic maps of the MP3D and Gibson datasets. The semantic map datasets are generated using Semantic MapNet [73] and computing ground truth potential functions. The area potential is calculated by grouping unexplored free cells into connected components and associating each component to the map frontiers. Then, for each frontier, the area potential is computed as the sum of areas of connected cells associated with the frontier,

normalized by the total free space. The object potential function depends on the geodesic distance between a frontier location, x, and the target object category, o_t.

$$U_t^o(o_t, x) = \max\left(1 - \frac{d_g(o_t, x)}{d_{max}}, 0.0\right) \tag{2.61}$$

where d_g is the geodesic distance between x and o_t, and d_{max} is the distance at which U_t^o decays to zero, which is chosen by validation experiments.

During evaluation, the two potential maps are linearly combined and the cell with the maximum potential is selected as the long term goal, which is reached following a fast-marching policy.

PONI reaches 0.200 success rate and 0.088 SPL on the 2021 Habitat challenge leaderboard.

Offline Visual Representation Learning

Offline visual representation learning (OVRL) learns visual representations on the Omnidata starter dataset [74] using a DINO [75] teacher-student network configuration. The learnt teacher network becomes the encoder layer of RGB observations, while depth observations are preprocessed by a frozen point goal navigation trained ResNet50 [2]. These encoded features, along with goal and pose information are forwarded through their respective linear network layers and fed to a recurrent GRU network. The output is fed through another fully connected layer which outputs the agent's actions. The agent is trained using imitation learning in a similar manner to Habitat-Web [33].

OVRL obtains 0.249 success rate and 0.083 SPL on the 2021 Habitat Challenge.

AllenAct ResNet18

The baseline agent from [76] uses a frozen, ImageNet trained ResNet18 network to preprocess agent RGB-D observations. A peculiar method of embedding depth information is achieved by tiling to form a 3 channel observation that is fed to the frozen preprocessing network. Notably, this approach does not rely on pose sensing or semantic segmentation.

The model receives a RGB image, the target object (i.e., the goal), and a depth image. The RGB image is encoded into a 512x7x7 tensor, I, by an ImageNet trained ResNet18 model without pooling and classification layers. The depth frame is tiled and is also fed to the frozen ResNet18 to form the tensor, D. The goal is used to index a trainable embedding matrix and form a 32-dimensional goal embedding G. I and D are then compressed by their respective two layer CNNs to form D' and I', each with dimensions 32x7x7. The vector G is tiled to a shape of 32x7x7 and concatenated with I' and D'. These two 64x7x7 tensors are passed through another two-layer CNN, each resulting in a 32x7x7 shape, and are concatenated. The result is passed through another two layer compressing CNN and flattened, forming a 1568 goal-conditioned visual embedding that is passed into a 1-layer, 512 dimensional GRU. The recurrent output is fed to the policy network $\pi_\theta(a_t|s_t)$ and value network $V_\phi(s_t)$ where θ and ϕ share parameters. Figure 2.3 illustrates the agent architecture.

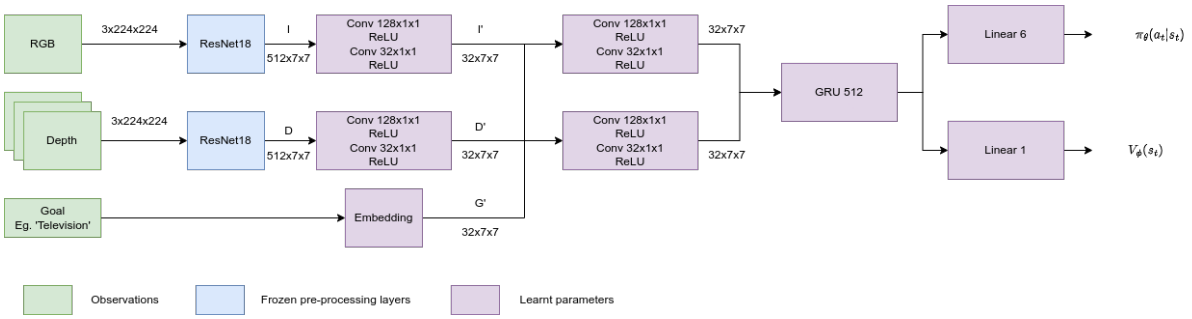

Figure 2.3: The AllenAct ResNet18 agent architecture illustrated[2].

The agent is trained using DD-PPO over multiple multi-GPU nodes using the Allen Act framework [76] and is able to achieve 0.1133 SPL and 0.2632 success rate on the 2021 RoboTHOR challenge test dataset. This is remarkably less than the performance on the validation set — roughly 10% less. This depicts a sizeable distribution difference between test and validation sets. In [5], where a similar method is used, discrepancies when using ImageNet trained networks in simulation versus their real world counter parts are illustrated via a dimensionality reduction evidencing a source of the sim-2-real gap.

EmbCLIP

Khandelwal et al. [34] use a similar model architecture for learning policies and value functions as AllenAct ResNet18. It differs in that a fixed CLIP ResNet50 [31] embedding network is used as a preprocessing layer, which extracts richer visual features compared to the ImageNet pretrained ResNet18. Moreover, this agent operates only using RGB sensory information.

[2]Retrieved from version v0.3.1 of [76]

The EmbCLIP agent obtains 0.2004 SPL and 0.4701 success rate on the 2021 RoboTHOR challenge, indicating how simply improving the richness of visual features can vastly improve performance. EmbCLIP also performs relatively well on the 2021 Habitat challenge (especially considering only RGB sensing is used) with an SPL of 0.078 and success rate of 0.181.

ProcTHOR

As is often noted in ObjectNav approaches, generalization error between training and validation scenes remain a barrier to achieving greater success rates and SPL scores in unseen environments. Deitke et al. [77] improve generalization error by procedurally generating 10000 environments by varying layouts, objects, lighting and materials in Unity [62]. They perform an ablation study evaluating the effect of increasing the training scenes and demonstrate state-of-the-art zero-shot test success rate and SPL on multiple THOR based challenges. It is worth noting that the limited, although still remarkable, zero-shot performance when transferring to the Habitat 2022 challenge is likely due to differing simulated environments; Habitat uses the Matterport 3D dataset while RoboTHOR and other THOR based challenges use Unity. The ablation study is in agreement with [78] where increasing training levels reduce generalization error. Performance is further improved after fine tuning the network on their respective challenge datasets. EmbCLIP agents are trained using DD-PPO as in [34] for 423 million steps during pre-training and 29 and 220 million steps during fine tuning for RoboTHOR and HM3D respectively.

The ProcTHOR agent once fine-tuned can obtain 0.6515 success rate and 0.2884 SPL

on the RoboTHOR challenge environment, and obtain 0.54 success rate and 0.32 SPL on the 2022 Habitat challenge leaderboard.

Summary

This section provided an overview of different approaches to the ObjectNav problem and common themes are apparent. A large portion of approaches to the Habitat challenge rely on semantic segmentation and, optionally, the construction of 2D semantic maps. Despite the appealing nature of incorporating priori knowledge into the pipeline of ObjectNav agents, it is unclear to what degree the predicted and ground truth semantic segmentation error will prove an obstacle. Errors in constructing 2D maps, semantic masks, and agent pose can also be challenging, especially considering that indoor pose estimation is a non-trivial problem in itself.

Reward schemes in all approaches commonly include a success term, $r_{success}$, to reward completing the task, and a slack term, r_{slack}, to encourage trajectory efficiency. The success term commonly ranges between 5 and 10 and is rewarded upon success, while r_{slack} is often between -0.01 and -0.0001 and is applied at each timestep. These values are small as the goal is not to completely inhibit exploration. The geodesic distance between the agent and target object (Δ_{geo}) is often included in the reward schedule to deal with reward sparsity in the ObjectNav task. The overall ObjectNav reward can be formulated as

$$r = r_{slack} + r_{success} - w_{geo} \cdot \Delta_{geo} \tag{2.62}$$

where w_{geo} weights the geodesic reward.

Certain challenges associated with ObjectNav are tackled in the approaches mentioned in this section. Feature richness is one challenge in which solutions attempt to extract rich features to be used when formulating a policy. Exploration is another challenging aspect to ObjectNav which is inherent due to sparse positive rewards. Finally, one of the main challenges in ObjectNav is generalization, be it to simulated or physical environments.

All methods described have some form of feature extraction from the raw sensory inputs. The approaches in [20, 21, 33, 38, 69, 71] rely on semantic segmentation and, optionally, 2D semantic maps or SGE. Yadav et al. [79] and Ramrakhya et al. [33] make use of pre-trained point goal navigation ResNet50 networks [2] while Khandelwal et al. [34] and Deitke et al. [77] use CLIP ResNet50 preprocessors.

A subset of reinforcement learning approaches recognize the sparsity of positive rewards in the ObjectNav task and employ different methods to mitigate the problem. Ye et al. [20] use both a count-based exploration reward with a tethered policy to encourage exploration along with auxiliary tasks. Maksymets et al. [21] employ the "ExploreTillSeen" reward scheme which computes the percentage change in the explored 2D semantic map as an exploration bonus. Ramrakhya et al. [33] also approach the sparse reward problem using behavior cloning.

Another subset of methods make note of the generalization gap between training and validation/test datasets. Maksymets et al. [21] highlight that geodesic rewards can overfit the agent to training scenes and that both data augmentation and feature dimensionality reduction improves generalization. ProcTHOR provides an ablation study similar to [78] highlighting zero-shot performance on challenge datasets after pre-training on varying numbers of procedurally generated indoor scenes. Each of these results highlight the

size limitations of challenge datasets.

Table 2.1 provides a brief overview of ObjectNav approaches while tables 2.2 to 2.5 compare performances on different ObjectNav challenges[3]. The top performers are also listed although their citations are absent. Note that comparisons between performances on different challenge datasets are not trivial given the differences in their respective simulated environments. For example, the target object classes and their initial distances from the agent vary greatly between RoboTHOR, MP3D and HM3D datasets.

[3]Performance entries were retrieved on 2022-09-01 from their respective online leaderboards:
Habitat 2020: https://aihabitat.org/challenge/2020/
Habitat 2021: https://eval.ai/web/challenges/challenge-page/802/leaderboard/2195
Habitat 2022: https://eval.ai/web/challenges/challenge-page/1615/leaderboard/3899
RoboTHOR: https://leaderboard.allenai.org/robothor_objectnav/submissions/public

Table 2.1: ObjectNav approach comparison table.

Method	Sensing	Preprocessing	Segmentation & Mapping	Planning	Training
SemExp [38]	RGB-D Pose Goal	—	Mask R-CNN 2D Occupancy	Target Point Goal Policy Network FMM [64]	Supervised PPO
RedRabbit [20]	RGB-D Pose Goal	—	RedNet SGE	Policy Network	RedNet fine-tune PPO w/ Aux-Tasks Tethered Policy
Stubborn [69]	RGB-D Pose Goal	—	RedNet SGE 2D Occupancy	Clockwise Explore FMM [64]	RedNet fine-tune w/ Naive Bayes
THDA [21]	RGB-D Pose Goal	—	RedNet	Policy Network	DD-PPO ExploreTillSeen Data Augmentation
Habitat-Web [33]	RGB-D Pose Goal	ResNet50 [2]	RedNet	Policy Network	Imitation Learning
PONI [71]	RGB-D Pose Goal	—	Mask R-CNN 2D Occupancy	Max-Potential Point Goal FMM [64]	Supervised
OVRL [79]	RGB-D Pose Goal	ResNet50 [2]	—	Policy Network	Imitation learning
AllenAct ResNet18 [76]	RGB Goal	ResNet18 [30]	—	Policy Network	DD-PPO
EmbCLIP [34]	RGB Goal	CLIP [31]	—	Policy Network	DD-PPO
ProcTHOR [77]	RGB Goal	CLIP [31]	—	Policy Network	DD-PPO Data Augmentation

Table 2.2: Habitat 2020 ObjectNav leaderboard.

Method	Test Success	Test SPL
SemExp [38]	0.25	0.10

Table 2.3: Habitat 2021 ObjectNav leaderboard.

Method	Test Success	Test SPL
IL-HD+RL-FT	0.337	0.146
Habitat-Web (IL-HD) [33]	0.278	0.099
Stubborn [69]	0.237	0.098
TreasureHunt [21]	0.214	0.089
PONI [71]	0.200	0.088
OVRL [79]	0.249	0.083
EmbCLIP [34]	0.181	0.078
RedRabbit [20]	0.237	0.062

Table 2.4: Habitat 2022 ObjectNav leaderboard.

Method	Test Success	Test SPL
Stretch (Semantic Map + Frontier)	0.60	0.34
IL-HD+RL-FT	0.65	0.33
ProcTHOR [77]	0.54	0.32
Habitat-Web (IL-HD) [33]	0.55	0.22

Table 2.5: RoboTHOR 2021 ObjectNav leaderboard.

Method	Test Success	Test SPL
ProcTHOR Fine Tune [77]	0.6515	0.2884
EmbClip [34]	0.4701	0.2004
AllenAct ResNet18 [76]	0.2632	0.1133

Chapter 3: End-to-End Object Goal Navigation with Proximal Policy Optimization

The previous chapter reviewed the literature relevant to reinforcement learning (RL) in object goal navigation (ObjectNav), and highlighted challenges that are faced when solving the task. Crucial problems to address include feature representation, generalization on test scenes, and sparse positive rewards. This chapter will empirically demonstrate these aforementioned issues in the RoboTHOR 2021 challenge with its corresponding baseline agent. Furthermore, it will present an ablation study to gain insight on components of the baseline, and propose modifications that improve success and trajectory efficiency in unseen environments while balancing the trade-off between training time and performance.

To this effect, end-to-end on-policy embodied agents are trained and evaluated on this ObjectNav task. The baseline AllenAct model is adopted initially to demonstrate the challenges faced in the task, and to propose a modified agent based on empirical evidence. Experiments are conducted varying the observation space, action space, normalization techniques, reward schemes, and training hyperparameters to evaluate the capabilities of RL trained agents during ObjectNav. The agents implement the ResNet18 baseline architecture from [76] since it does not rely on semantic segmentation nor pose estimations, which are subject to noise in physical environments.

Although the ResNet18 features are not without their own limitations in regard to their utility in ObjectNav — as will be highlighted in this chapter — they are useful in reducing observation dimensionality in a lightweight manner, improving memory usage and training time.

Section 3.1 provides an analysis on the RoboTHOR challenge dataset and its object class distributions across episodes. Section 3.2 details the ResNet18 AllenAct agent and identifies limitations, particularly in a short run of 10 million steps. Section 3.3 presents an ablation study of the different components of the baseline agent and proposes a modified agent that improves validation metrics and reduces the generalization barrier present in this task, while also constraining training time. The resulting agent is trained with the full 6-action space, red-green-blue (RGB) observations, no geodesic dense reward, and 50M environment interactions.

3.1 RoboTHOR 2021 Dataset

As mentioned in section 2.3.1, the RoboTHOR 2021 challenge dataset is constructed from examples of real apartments and replicated in the Unity real-time game engine [62]. The modalities available to the LoCoBot agent in this simulated environment are RGB-D. Additionally, RoboTHOR defines 12 target object classes and consists of 89 scenes which are split into 60 train, 15 validation, 10 standard test scenes and 4 development test scenes. Episodes in this dataset have a maximum duration of 500 environment interactions and consist of a starting point in a scene, target object and a shortest path length.

The simulated LocoBot agent takes discrete actions such that move forward actions displace the robot in steps of 0.25m, and rotation actions rotate the agent in steps of 30°. These step sizes are stochastic and the environment is continuous. That is, the agent does not snap to positions on a grid of the scene.

In this thesis, experiments are conducted on the training and validation sets since

the test sets are used for submissions to the RoboTHOR leaderboard[1]and do not allow for evaluating success within the simulated environment. Episodes are evenly split per object, and episode difficulty is established based on the 20th and 60th percentiles of shortest path lengths between the target objects and the agent's initial positions. That is, the scenarios with the top 20% of shortest path lengths to target objects are considered "easy", the scenarios between the 20th and 60th percentiles of shortest paths are considered "medium", and the episodes within the 60th and 100th percentiles of shortest path lengths are considered "hard".

Although the overall dataset may be split evenly over all objects, it is worth determining if there is an underlying bias towards some objects being "easier" to learn. This bias may present itself in the shortest path lengths per object class in the training dataset, or in the number of confounding goal classes in close proximity of the target. A target class with shorter path lengths or with less confounding objects near it is expected to be "easier" to learn.

The difficulty per target class for the training dataset is detailed in table 3.1 and indicates a variance between different object classes. Additionally, table 3.2 shows the number of episodes with confounding goal objects in close proximity of the target.

The two tables seem to suggest that the Television is the easiest target to learn, but it is unclear for the other classes how much one factor might impact training over another. It is clear that the object classes are not evenly distributed with respect to shortest path lengths and the frequency of confounding targets in proximity. This is likely a characteristic of

[1]The RoboTHOR leaderboard can be found at the following link: https://leaderboard.allenai.org/robothor_objectnav/submissions/public

ObjectNav tasks given that several target objects are more likely to be found together. For example, smaller goals are often found in proximity of a desk, whereas garbage cans are frequently placed on the floor in isolation.

Table 3.1: Occurrences of target objects and episode difficulties.

Difficulty	Apple	AlarmClock	Vase	Bowl	SprayBottle	BasketBall	GarbageCan	Television	Mug	Laptop	HousePlant	BaseballBat
Easy	1722	1612	1755	1890	1588	1785	1590	1983	1767	1882	1821	1719
Medium	3673	3914	3875	3523	3688	3731	3445	3523	3649	3389	3629	3647
Hard	3605	3474	3370	3587	3724	3484	3965	3494	3584	3729	3550	3634

Table 3.2: Occurrences of confounding target objects in proximity (0.2m).

Object	Apple	AlarmClock	Vase	Bowl	SprayBottle	BasketBall	GarbageCan	Television	Mug	Laptop	HousePlant	BaseballBat
Apple	0	1800	1800	0	0	0	0	0	5400	0	1800	0
AlarmClock	1800	0	0	0	0	0	0	0	0	0	0	0
Vase	1800	0	0	0	0	0	0	0	5400	0	0	0
Bowl	0	0	0	0	3600	0	0	0	3600	0	0	0
SprayBottle	0	0	0	3600	0	0	0	0	0	0	1800	0
BasketBall	0	0	0	0	0	0	0	0	0	0	0	0
GarbageCan	0	0	0	0	0	0	0	0	0	0	0	0
Television	0	0	0	0	0	0	0	0	0	0	0	0
Mug	5400	0	5400	3600	0	0	0	0	0	0	1800	0
Laptop	0	0	0	0	0	0	0	0	0	0	0	0
HousePlant	1800	0	0	0	1800	0	0	0	1800	0	0	0
BaseballBat	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.1 provides samples of the object classes encountered in the RoboTHOR dataset and shows some examples of confounding objects in proximity.

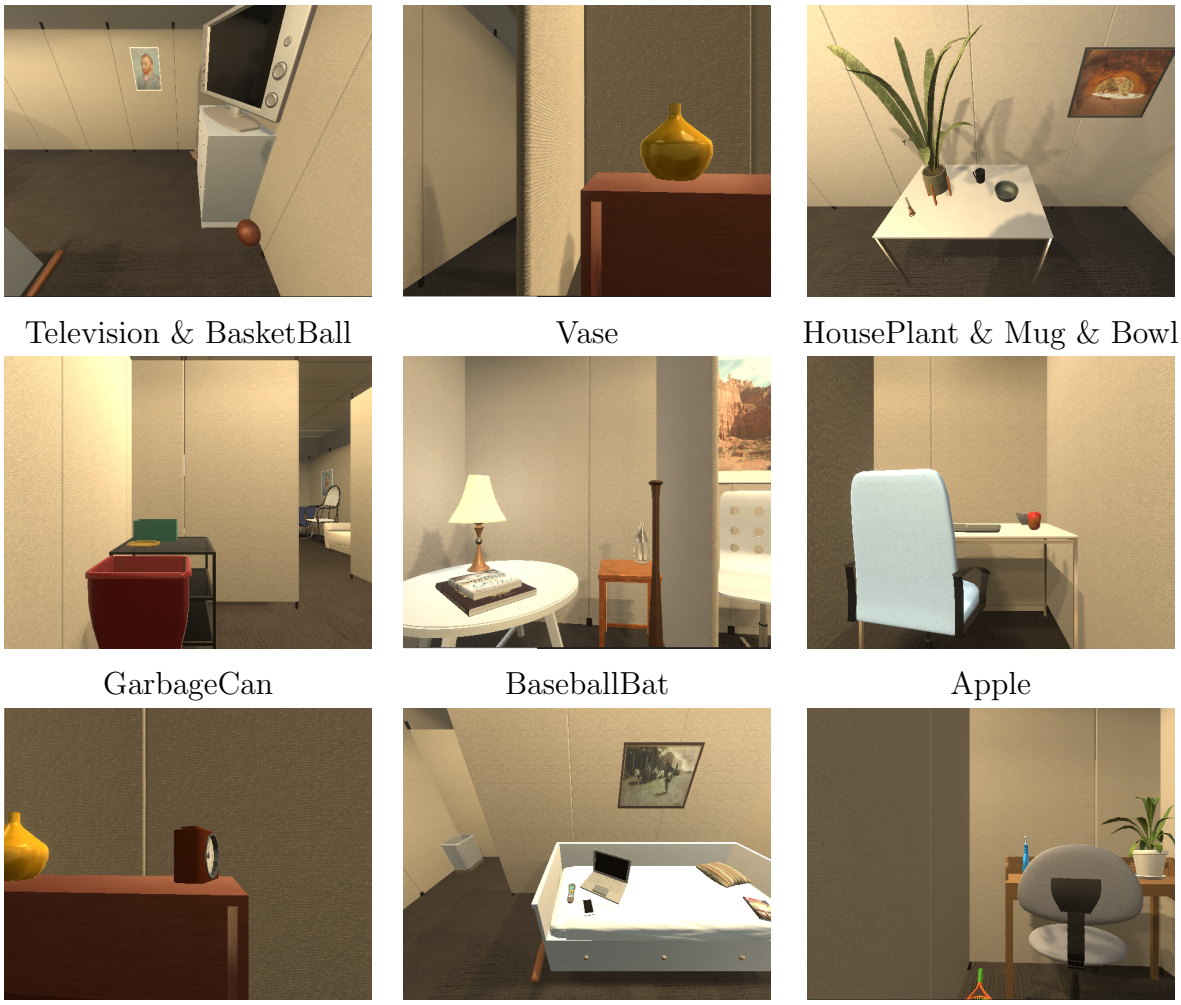

Figure 3.1: RoboTHOR object class examples

Figure 3.2 illustrates the distribution of shortest path lengths for the training and validation datasets of the RoboTHOR challenge for the HousePlant and AlarmClock object classes. In the case of the HousePlant class, the validation dataset contains a higher percentage of scenarios with short path lengths when compared to the training dataset.

In contrast, the reverse is true for the AlarmClock class — it is the training dataset that contains a greater proportion of episodes with short path lengths. These discrepancies can contribute to greater generalization errors between training and validation scenarios. However, Deitke et al. [77] demonstrate that this problem can be mitigated by pretraining an ObjectNav agent in a dataset two orders of magnitude larger than the RoboTHOR training dataset.

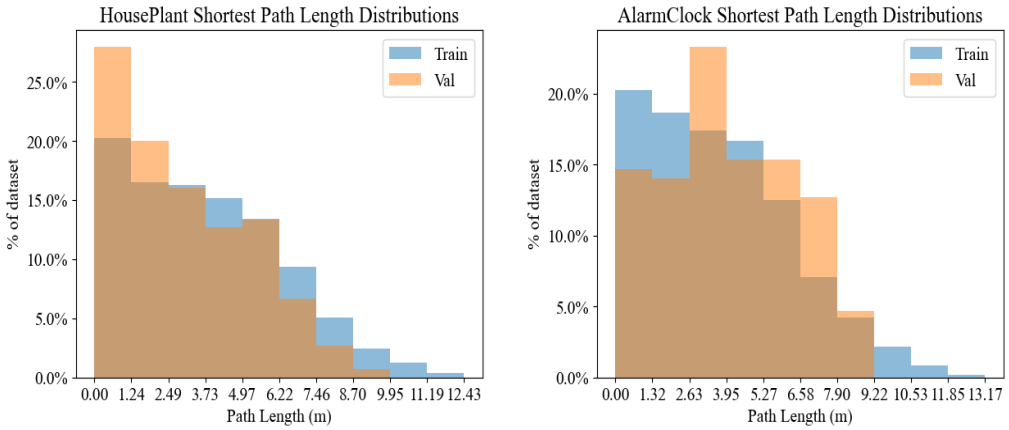

Figure 3.2: Distribution of episode shortest path lengths in the RoboTHOR training and validation datasets for the HousePlant (left) and AlarmClock (right) object class.

3.2 AllenAct ResNet18 Baseline

The AllenAct ResNet18 [76] agent is selected as the baseline for the RoboTHOR challenge. It led the corresponding leaderboard until EmbCLIP [34], which simply replaces the ResNet18 feature extraction layer with a CLIP ResNet50 [31]. The EmbCLIP agent is not used as the baseline for the experiments presented in this thesis since it was published in

late 2021 to 2022. By this time, most of the experiments for this thesis had already been completed.

3.2.1 The Original Baseline

The AllenAct ResNet18 baseline observes RGB and tiled depth frames that have been resized, cropped and normalized using the ImageNet dataset's [30] RGB means and standard deviation. It is worth noting that this method of pre-processing depth frames is unusual and the depth streams omission is investigated in section 3.3.2. The RGB and depth observations are then processed by the ResNet18 [26] feature extraction layer which outputs two 512x7x7 tensors, I and D, while the goal is embedded into a 32x7x7 dimensional embedding, G. Both I and D are fed through CNN blocks and concatenated with G. Both streams are then passed through another convolutional block before being concatenated. The resulting tensor is fed through another convolutional block followed by a GRU network. This final tensor is fed to the policy and value function networks. Further details on the network architecture can be found in section 2.3.3.

The baseline agent referenced in the AllenAct leaderboard[2] is trained using decentralized distributed proximal policy optimization (DD-PPO) [2], a variant of proximal policy optimization (PPO) that scales up across multiple GPU nodes in a distributed system. PPO is an on-policy RL algorithm that has been demonstrated to obtain impressive performances on a range of tasks while attaining a shorter wall-clock time relative to other RL

[2]The AllenAct ResNet18 baseline referred to in this thesis is from https://leaderboard.allenai.org/robothor_objectnav/submissions/public and the implementation details refer to version v0.3.1. [76].

algorithms. The PPO algorithm is described by algorithm 2.3 in appendix B.1, however several variations are commonly seen across its implementations. Namely, value loss and gradient clipping, entropy loss bonuses, and advantage normalization are all methods often applied to PPO. Gradient and loss clipping along with advantage normalization are used to stabilize training, while entropy bonuses encourage policy exploration. The version of PPO used in the original baseline makes use of these techniques; however, Andrychowicz et al. [80] empirically demonstrate that the aforementioned normalization method might not provide improvement.

The training algorithm uses normalized advantages which are estimated via generalized advantage estimation (GAE) [52]. Advantages are computed as

$$A_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V \tag{3.1}$$

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{3.2}$$

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} \dots) \tag{3.3}$$

$$= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{3.4}$$

where $A_t^{GAE(\gamma,\lambda)}$ is the generalized advantage estimate, δ_t^V is the temporal difference (TD) residual, r_t is the reward function (equation (3.7)), $A_t^{(k)}$ is the k step discounted sum of temporal difference (TD) residuals, $V(s_t)$ is the value prediction, γ is the discount factor, λ is the advantage exponential weight coefficient, and t denotes the timestep. This formulation of the advantage resembles TD(λ) value function estimates. The policy and

overall loss functions become

$$\mathcal{J}_\pi(\theta) = \min\left(\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})} A_t, clip\left(\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})}, 1-\epsilon, 1+\epsilon\right) A_t\right) \tag{3.5}$$

$$\mathcal{J}(\theta, \theta_v) = \mathbb{E}\left[\mathcal{J}_\pi(\theta) - c_V(V(s_t) - \hat{V}(s_t, \theta_v))^2 + c_e H\right] \tag{3.6}$$

where π is the policy action distribution, θ and θ_v denote the policy and value function parameters respectively, ϵ is the clipping coefficient, A_t is the generalized advantage estimate, $V(s_t)$ is the value function target (calculated as the discounted return), $\hat{V}(s_t)$ is the value function prediction, c_V is the value loss coefficient, c_e is the entropy bonus coefficient, and H is the entropy of agent action distributions.

The baseline reward schedule used is

$$r = r_{slack} + Sr_{success} - w_{geo} \cdot \Delta_{geo} \tag{3.7}$$

where $r_{slack} = -0.01$ at each timestep, $r_{success} = 10$, S $\in \{0, 1\}$ indicates success, $w_{geo} = 1$ and Δ_{geo} is the change in geodesic distance from the target object at each environment interaction.

3.2.2 Alterations to the Baseline

The baseline PPO configuration used in this thesis follows the configuration of the AllenAct ResNet18 baseline [76], with minor modifications to adapt training to a single workstation with fewer GPU resources. The main alteration compared to the original baseline pertains to the different version of PPO used; the original baseline uses DD-PPO across multiple

GPU nodes, which hastens training time, while the experiments presented in this thesis use PPO (i.e., without the decentralized distributed implementation).

The PPO configuration details and hyperparameters used can be found in table 3.3. The agent is trained for 10M environment interactions. Although this is a relatively short number of steps compared to the original AllenAct baseline, the performance in the short run can still demonstrate challenges encountered in ObjectNav and provide a baseline to improve upon. The rollout size, batch size, minibatch size, and number of parallel workers are adapted to suit the hardware used to train the ObjectNav agent. The rest of the PPO hyperparameters are kept the same as the original baseline.

Table 3.3: PPO hyperparameters.

Hyperparameter	Value
Learning rate	3e-4
Learning rate decay	[3e-4, 3e-6]
Discount factor γ	0.99
GAE λ	0.95
# SGD epochs	4
Value function coefficient	0.5
Entropy coefficient	0.01
PPO clipping	0.1
Gradient clipping	0.5
Value function clipping	0.1
Rollout size	258
Batch size	8192
Minibatch size	512
Advantage normalization	On
Max Interactions	10M
# parallel workers	32

3.2.3 Experiments

The ResNet18 agent is trained over roughly 10 million (10002432 since environment interactions are a multiple of the batch size hyperparameter) interactions in the RoboTHOR 2021 challenge training dataset and evaluated on the validation set following the configuration described in section 3.2. The experiments are run on a 32 core, 64 thread workstation with an Nvidia RTX 2080 graphical processing unit (GPU) for environment simulation and an RTX 2060 SUPER for agent model inference and backpropagation on losses.

The Ray RLlib [81] v1.2 framework is used to train the ObjectNav agent and the RoboTHOR challenge environment is adapted to the OpenAI Gym framework [82]. This enables efficient parallelization across actors and allows for the customization of RL algorithms within the framework. Additionally, the framework is flexible in using different neural network backend libraries like PyTorch [83] and Tensorflow [84].

Each training iteration of PPO collects a batch of trajectory information that is composed of one rollout per parallel worker. Rollouts can contain multiple full or incomplete trajectories that amount to a predefined size of environment interactions. Hence, the batch size is a multiple of the rollout size, and the total number of environment interactions collected are a multiple of the batch size. During backpropagation, the overall batch size is split into mini-batches for training on GPUs with an Adam optimizer [44]. After each training iteration, the learning rate is decayed using a linear schedule between maximum and minimum learning rates. Checkpoints are taken during training and later tested on the validation set.

Figure 3.3 demonstrates the training dynamics over roughly 10 million environment in-

teractions along with validation performances on agent model checkpoints taken after the 500th, 1000th, and final training iterations. Note that an iteration involves training the agent on a batch size (8192 in this case) of environment interactions; hence, the checkpoint at iteration n corresponds to the checkpoint after n batches of environment interactions. As mentioned in section 2.3.2, the metrics tracked are the success rate and success weighted by path length (SPL), measuring the agent's ability and efficiency in completing the ObjectNav task. The figure shows that the validation success curve steadily increases to 12.33%. Hence, validation performance may still increase with continued training.

It should be noted that there is an interplay between the learning rate decay and the number of interactions during training. Section 3.3.4 demonstrates this phenomenon when increasing the number of environment interactions used for training. It should also be noted that the training metrics reported relate to the training iteration's batch of environment interactions on a subset of training scenes, while the validation metrics correspond to performances on the entirety of the validation dataset.

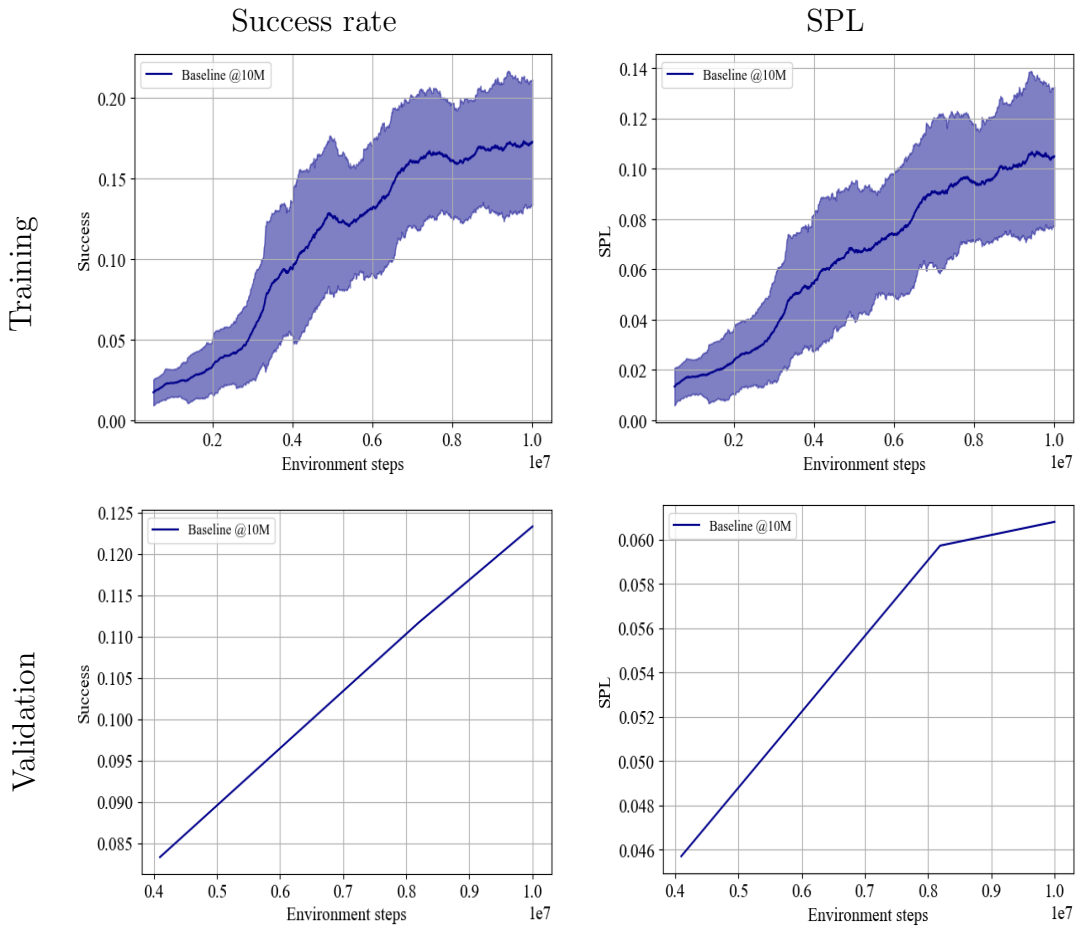

Figure 3.3: Training and validation curves of the baseline agent trained for 10M environment interactions.

Table 3.4 shows that the Television and HousePlant target classes were learnt almost to completion during training, while the remaining objects were almost entirely unsuccessful. As highlighted in section 3.1, the dataset seems to have a bias towards the "easier" Television episodes; although HousePlant episodes did not clearly show this same level of bias. Hence, the performance in both these classes are not attributable solely to the RoboTHOR

dataset's object distributions across episodes.

Table 3.4: Performance metrics of the baseline agent trained for 10M environment interactions.

Object	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.0000	0.0000	0.0133	0.0130
Apple	0.0000	0.0000	0.0200	0.0193
BaseballBat	0.0714	0.0714	0.0267	0.0233
BasketBall	0.0345	0.0345	0.0200	0.0198
Bowl	0.0000	0.0000	0.0133	0.0133
GarbageCan	0.0278	0.0278	0.0467	0.0404
HousePlant	1.0000	0.5683	0.6000	0.2595
Laptop	0.0000	0.0000	0.0333	0.0333
Mug	0.0000	0.0000	0.0467	0.0389
SprayBottle	0.0000	0.0000	0.0067	0.0067
Television	0.9348	0.5373	0.6333	0.2487
Vase	0.0000	0.0000	0.0200	0.0133
Difficulty	Train success rate	Train SPL	Val success rate	Val SPL
Easy	0.3382	0.1313	0.2555	0.1209
Medium	0.2171	0.1318	0.1104	0.0530
Hard	0.1773	0.1366	0.0614	0.0353
All episodes	0.2249	0.1337	0.1233	0.0608

It is possible that the difference in object performances could be a consequence of the ImageNet trained ResNet18 feature extraction layer. The ImageNet dataset has roughly 1.4M images pertaining to 1000 classes, which include several varieties of plant species and ball types but do not include baseball bats. EmbCLIP [34] has already shown that using a similar agent with a CLIP ResNet50 feature extraction module instead of the ImageNet trained ResNet18 performs much better on the RoboTHOR challenge 2021 dataset. This shows that even though the feature representation models share a similar network architecture, one enables much better transfer learning for the ObjectNav task than the other. Although the performance of the EmbCLIP model is improved with CLIP [31] feature representation, the scope of this thesis focuses on evaluating solutions to the sparsity of rewards in ObjectNav against an appropriate baseline.

Television and HousePlant are the classes which drive overall performance and demonstrate validation success rates of 63% and 60% respectively. The gap between their validation and training success rates are roughly 30% and 40%, respectively. This suggests that, once the agent has successfully learnt to find the object in the training set, the generalization error still remains large.

Table 3.4 also shows that a larger distance between the target and the agent's initial position ("higher difficulty") leads to lower success rates in the training and validation sets. However, the training SPL remains mostly constant as the difficulty increases, which suggests that the agent is not finding the target objects via an exploratory policy but rather by memorizing paths. In contrast, validation SPLs and success rates decrease as difficulty increases, which is expected due to the impact of sparse rewards.

Finally, the generalization gap between training and validation metrics can be extracted from table 3.4 and computed, as mentioned in section 2.3.2. Success rate generalization error reaches 8.27%, 10.66%, and 11.59% across easy, medium and hard difficulties, respectively. SPL generalization errors show greater sensitivity to episode difficulties, with rates of 1.04%, 7.88%, and 10.13% for easy, medium and hard difficulties, respectively. Larger generalization errors point to the agent overfitting to training episodes.

3.3 Baseline Ablation Study and Proposed Agent

Given the limitations faced in the original baseline, as discussed in section 3.2.3, an ablation study of different components of the baseline is presented in this section to provide insight

into their effects on validation performance metrics. This includes an analysis of each component's effect on object class and episode difficulty performance metrics. The empirical observations in this section guide the design choices made when proposing an agent that shows better success and trajectory efficiency in unseen environments (via validation success rate and SPL metrics) while reducing inference and training time.

Embodied agents are trained varying the reward scheme, agent observations, action space, total training environment steps, and advantage normalization. The latter two components may be considered generally related to the RL algorithm used, whereas the former three are specific to the ObjectNav problem. To this effect, section 3.3.1 evaluates the impacts of removing geodesic dense rewards from the reward scheme, section 3.3.2 investigates RGB only sensing, section 3.3.3 explores whether 4-action agents can improve performance by reducing the tasks complexity, section 3.3.4 demonstrates how learning rate annealing influences learning dynamics, and section 3.3.5 evaluates the benefit of advantage normalization for this task. Finally, section 3.3.6 formally describes the proposed agent architecture and section 3.3.7 evaluates that agent against a baseline that is trained for an equal number of environment interactions.

As mentioned in section 2.3.2, training and validation metrics are shown for success rate and SPL. In addition, tables show performance metrics across different target objects and episode difficulties, and bar charts provide easy comparisons to other agents presented in this chapter. Comparing an agent's performance over target objects provides some insight on the number of objects learnt, while comparisons made with respect to episode difficulty evaluate the effects of reward sparsity on the respective agents.

The experiments follow the same setup mentioned in section 3.2.3, and alterations made

to the baseline configuration are adopted serially, such that each successive experiment builds on those that are presented before it. The reader is to assume that the agents presented in this section use the baseline configuration described in section 3.2.3 unless otherwise specified. That is, the use of geodesic rewards, RGB-D observations, a full 6 action space, and advantage normalization. The proposed agent in section 3.3.6 is trained using PPO for 50M steps without geodesic rewards, without depth sensing, and without advantage normalization, and uses the default 6-dimensional action space. As of writing this thesis, the literature has not provided a demonstration of such a configuration on an ObjectNav task.

3.3.1 Geodesic Reward Removal

Maksymets et al. [21] highlight that the geodesic dense reward (Δ_{geo} in equation (3.7)) results in poor generalization when applied to ObjecNav. This reward, which penalizes actions that increase the distance to the target object, provides an inhibitory signal when exploratory policies are followed. This is despite the use of a slack penalty (r_{slack} in equation (3.7)), which also penalizes the agent's actions at each timestep to encourage efficiency in the agent's trajectory. As evidenced by the human benchmark obtained in [33], humans obtain a SPL of roughly half the success rate, highlighting the exploratory nature of the problem. As such, the reward schedule is modified by removing the geodesic reward while maintaining other hyperparameters constant, in order to evaluate whether such a bonus is necessary or beneficial.

Figure 3.4 illustrates the overall training dynamics and validation performance at three

model checkpoints for baseline agents trained with and without geodesic dense rewards. Checkpoints are relatively infrequent since inference on the whole validation dataset is time consuming, which explains the inflection points in the validation curves. The results suggest that training without geodesic rewards leads to similar training metrics by the end of training, but reduces generalization error.

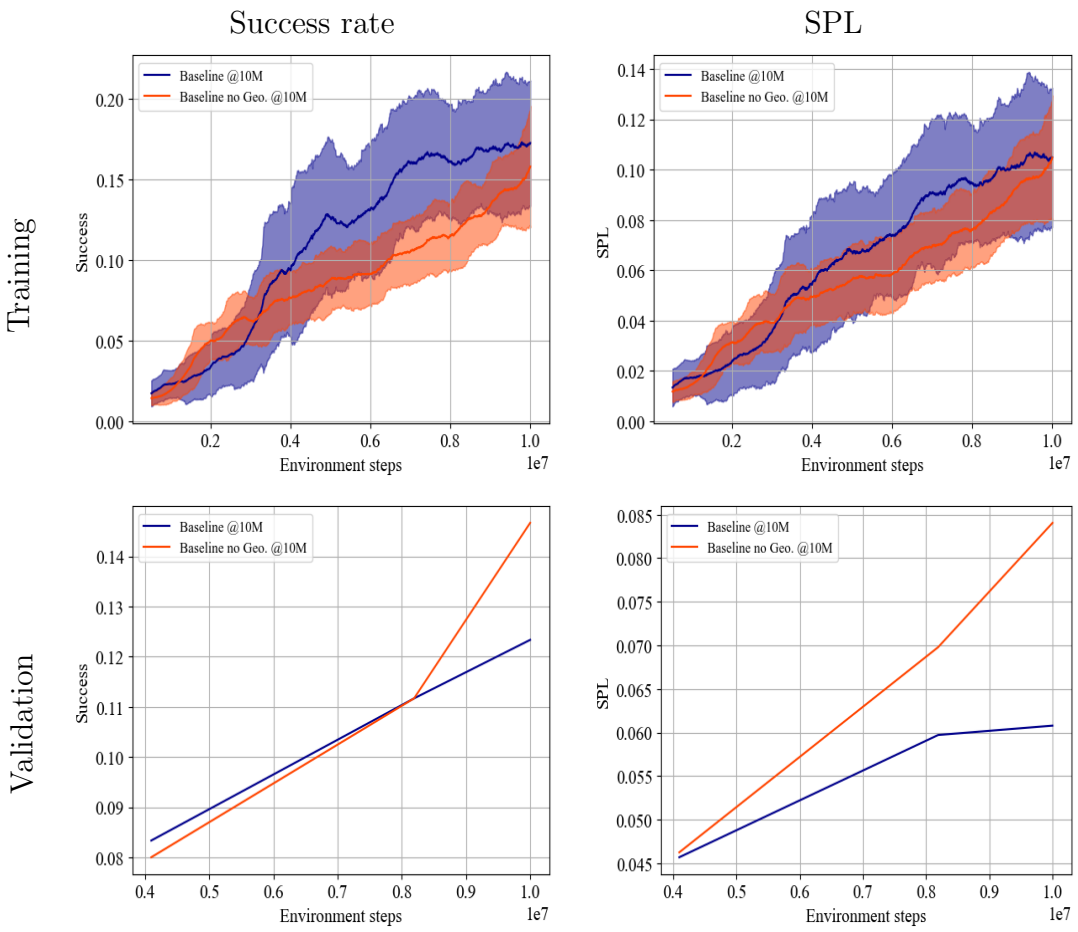

Figure 3.4: Training and validation curves of the baseline agent trained with and without dense geodesic rewards for 10M environment interactions.

Table 3.5 shows performance metrics across target classes and difficulties. Table 3.4 previously showed an overall generalization gap of 10.16% for the baseline agent. Now, table 3.5 shows the baseline agent trained without geodesic rewards reduces the gap to 6.45%. Table 3.5 also shows that the success rate generalization gap still persists for learnt objects; however the agent now partially learns other objects (e.g. BasketBall and GarbageCan) instead of memorizing the Television and HousePlant, as observed section 3.2.3.

Table 3.5: Performance metrics of the baseline agent trained without the geodesic reward for 10M environment interactions.

Object	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.0513	0.0256	0.0200	0.0067
Apple	0.0588	0.0441	0.0867	0.0690
BaseballBat	0.1364	0.1235	0.0200	0.0043
BasketBall	0.4000	0.2504	0.2400	0.1367
Bowl	0.0500	0.0454	0.0733	0.0518
GarbageCan	0.2955	0.1806	0.2067	0.1461
HousePlant	0.7660	0.5130	0.4400	0.2217
Laptop	0.0732	0.0616	0.1133	0.0746
Mug	0.0962	0.0925	0.0600	0.0511
SprayBottle	0.0189	0.0171	0.0267	0.0244
Television	0.6364	0.3624	0.4533	0.2058
Vase	0.0566	0.0314	0.0200	0.0165
Difficulty	Train success rate	Train SPL	Val success rate	Val SPL
Easy	0.5308	0.3135	0.4313	0.2247
Medium	0.1888	0.1429	0.1104	0.0713
Hard	0.0381	0.0311	0.0249	0.0167
All episodes	0.2112	0.1406	0.1467	0.0841

Figures 3.5 and 3.6 compare validation metrics of the baseline agents trained with and without geodesic rewards. Figure 3.5 shows that the validation success rates with Television and HousePlant classes drop significantly, likely due to decreased success in hard scenarios. Figure 3.6 shows that omitting the geodesic reward when training the AllenAct baseline results in more success in episodes with shorter path lengths to the target object (i.e lower level of difficulty), but achieves worse metrics for more difficult episodes. Validation SPL

in easy and medium episodes also improved, likely due to the agent successfully learning to find targets in close proximity without much exploration.

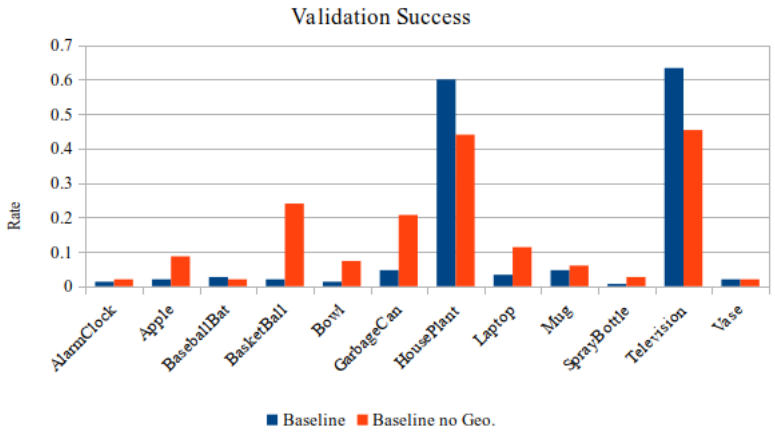

Figure 3.5: Object class validation success rate comparison between baseline agents trained with and without geodesic rewards.

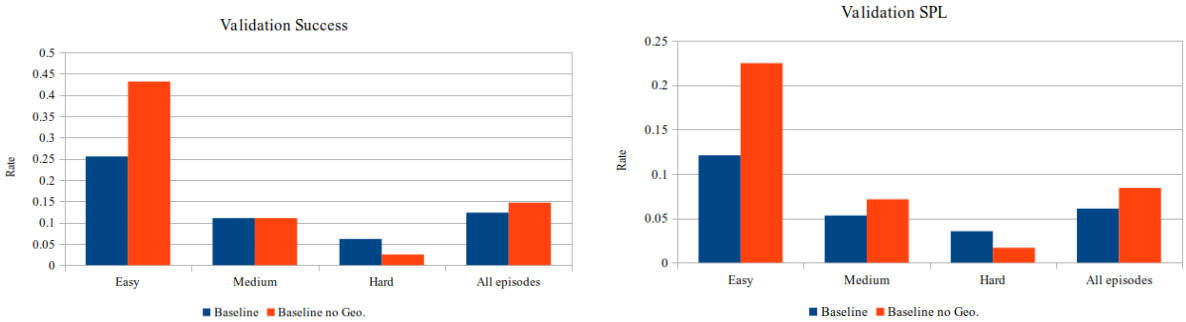

Figure 3.6: Episode difficulty validation metrics comparison between baseline agents trained with and without geodesic rewards.

Since removing the geodesic reward improves overall validation metrics, the agent proposed in section 3.3.6 drops the geodesic reward. The experiments that follow in this ablation study also do not use geodesic rewards.

3.3.2 RGB-D vs RGB

As shown in figure 2.3, the observation processing step of the baseline AllenAct agent tiles depth frames into three channels and maps the result as an RGB tensor via the ImageNet trained ResNet18 model. Refer to section 2.3.3 for a detailed description of the AllenAct ResNet18 agent. This process applied to depth frames is suspicious, as the ResNet18 model was never trained on depth images and the pixel distribution per image varies greatly from depth sensing. In contrast, the EmbCLIP agent [34] does not make use of depth sensing.

As such, the benefit of depth observations are explored in this thesis by experimenting with RGB-only agents, potentially reducing training and inference time by not rendering depth frames. The agent's architecture is trimmed down, removing the depth stream and its network relevant components, as illustrated in figure 3.7. This agent architecture is hereon referred to as the RGB ResNet18.

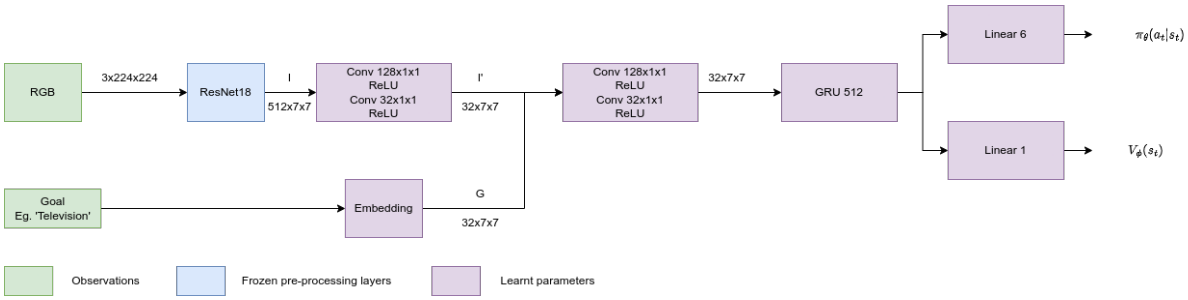

Figure 3.7: The RGB ResNet18 agent architecture.

Figure 3.8 compares the training and validation curves of agents trained with and without the depth modality (and no geodesic rewards as per section 3.3.1). The agent endowed with depth (Baseline no Geo.) achieves slightly greater training metrics and

validation success throughout training, but obtains slightly lower validation SPL after 10M environment interactions. The curves also demonstrate an increasing trend in both training and validation metrics, suggesting further training could improve each agent's results for each metric.

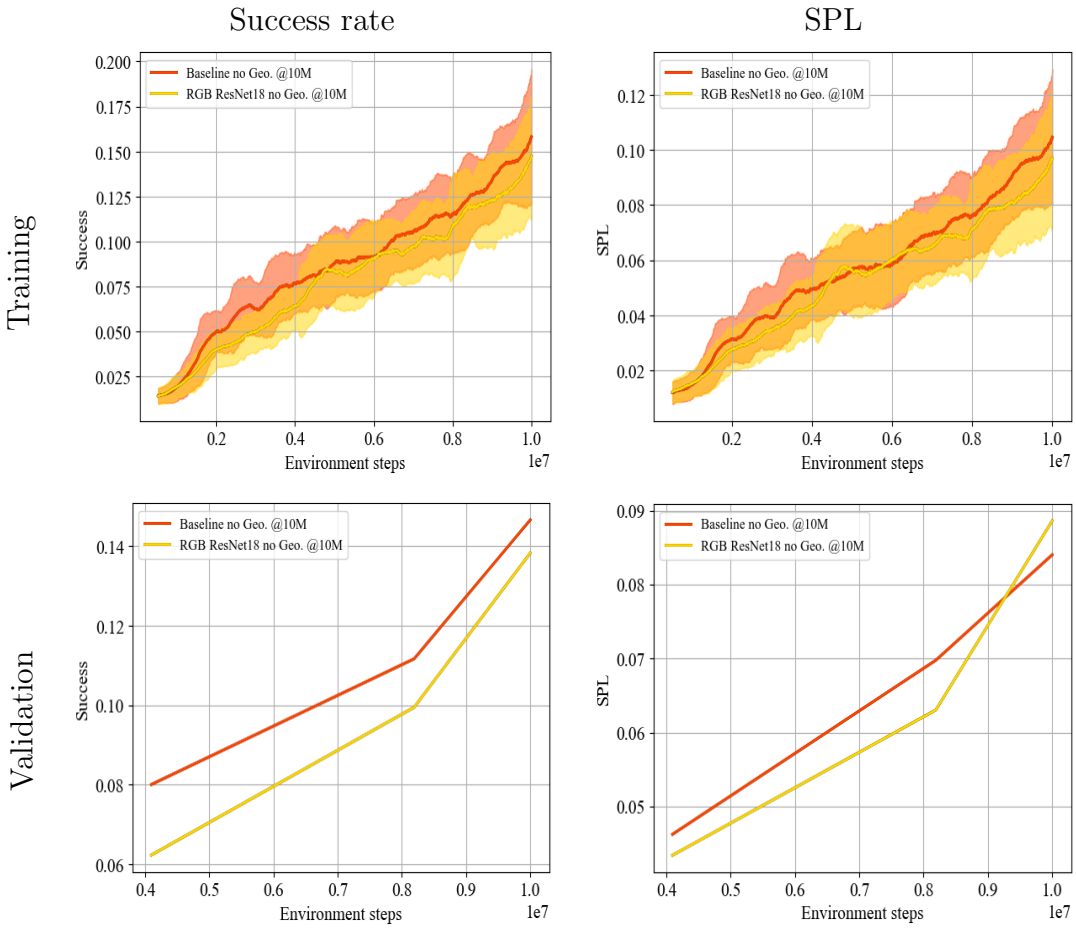

Figure 3.8: Training and validation curves of baseline and RGB ResNet18 agents trained without geodesic rewards for 10M environment interactions.

Table 3.6 reports on the depth-absent agent's performance across difficulties and ob-

ject classes. The agent obtains a 5.85% overall success generalization gap, which is not significantly different from the 6.45% its RGB-D counterpart obtained (from table 3.5).

Table 3.6: Performance metrics of RGB ResNet18 agent trained without the geodesic reward for 10M environment interactions.

Object	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.0851	0.0697	0.0133	0.0067
Apple	0.1818	0.1458	0.1600	0.1144
BaseballBat	0.0455	0.0324	0.0267	0.0241
BasketBall	0.3000	0.2020	0.2200	0.1070
Bowl	0.1053	0.0849	0.0533	0.0515
GarbageCan	0.2895	0.1843	0.2333	0.1582
HousePlant	0.5111	0.3542	0.3533	0.1906
Laptop	0.0811	0.0590	0.1200	0.1129
Mug	0.0286	0.0286	0.0733	0.0633
SprayBottle	0.0909	0.0859	0.0267	0.0221
Television	0.4706	0.3098	0.3267	0.1698
Vase	0.0652	0.0652	0.0533	0.0433
Difficulty	**Train success rate**	**Train SPL**	**Val success rate**	**Val SPL**
Easy	0.5000	0.3181	0.4038	0.2363
Medium	0.2105	0.1662	0.1140	0.0837
Hard	0.0262	0.0224	0.0116	0.0063
All episodes	0.1968	0.1409	0.1383	0.0887

Figures 3.9 and 3.10 compare validation metrics of the baseline agent trained with and without geodesic rewards, as well as the RGB ResNet18 agent trained without geodesic rewards. Figure 3.9 shows the RGB-only agent's validation success decreaces significantly on the HousePlant and Television classes while improving success on the Apple class. Differences in other object classes are marginal when compared to the RGB-D agent trained without geodesic rewards. Figure 3.10 shows the RGB-only agent is less successful but more trajectory efficient (i.e. greater SPL) in medium and easy episodes. That said, the performance differences between RGB and RGB-D agents trained without geodesic rewards with respect to the target class or episode difficulty could be attributed to the variance between runs. When comparing the training times of both actors (20.58h for the

RGB agent vs 34.33h for the RGB-D agent), the 67% increase in training time outweighs the slight overall validation success improvements demonstrated by including the depth stream. Furthermore, the RGB agent is lighter — that is, it has fewer hyperparameters and has lower memory requirements as a consequence of omitting depth observations and the corresponding depth processing stream of the agent. As a result, the depth modality is dropped from subsequent experiments, including the proposed agent in section 3.3.6.

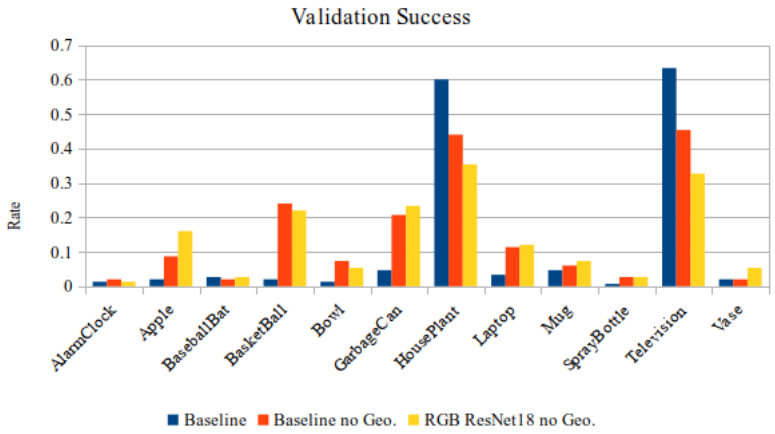

Figure 3.9: Object class validation success rate comparison between baseline, baseline no Geo., and RGB ResNet18 no Geo. agents.

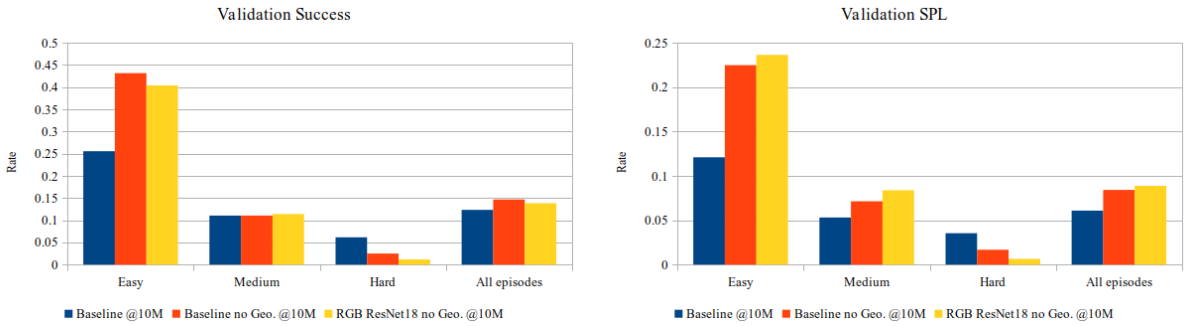

Figure 3.10: Episode difficulty validation metrics comparison between the baseline, baseline no Geo., and RGB ResNet18 no Geo. agents.

3.3.3 Reduced Action Space

Reducing an agents action space can improve the performance and wall clock training time in RL tasks by reducing their complexity. In ObjectNav, these benefits can be achieved as long as an agent can successfully view goal objects with a fixed camera horizon. Ye et al. [20] experiment with 4-action agents in the Habitat 2021 challenge and obtain agents that can perform well; however, policies that are allowed the full action space obtain better performances on validation and test sets. Nonetheless, the potential benefit in the RoboTHOR dataset remains to be evaluated; hence experiments are conducted in this thesis removing the vertical camera rotation actions (*look-up 30°* and *look-down 30°*) from the default 6-action agent, and fixing the camera horizon of the agent parallel to the ground at 0°. As mentioned in section 2.3.1, the original full action space is: *move-forward 0.25m, turn-left 30°, turn-right 30°, look-up 30°, look-down 30°, stop.*

Figure 3.11 illustrates the training and validation performance of 4 and 6-action RGB ResNet18 agents trained without geodesic rewards. Although the 4-action agent's training curve shows improved training success, it shows marginal improvements in validation, indicating a greater generalization gap. Furthermore, the 4-action agent's lower final validation SPL indicates the agent is less efficient in its trajectories.

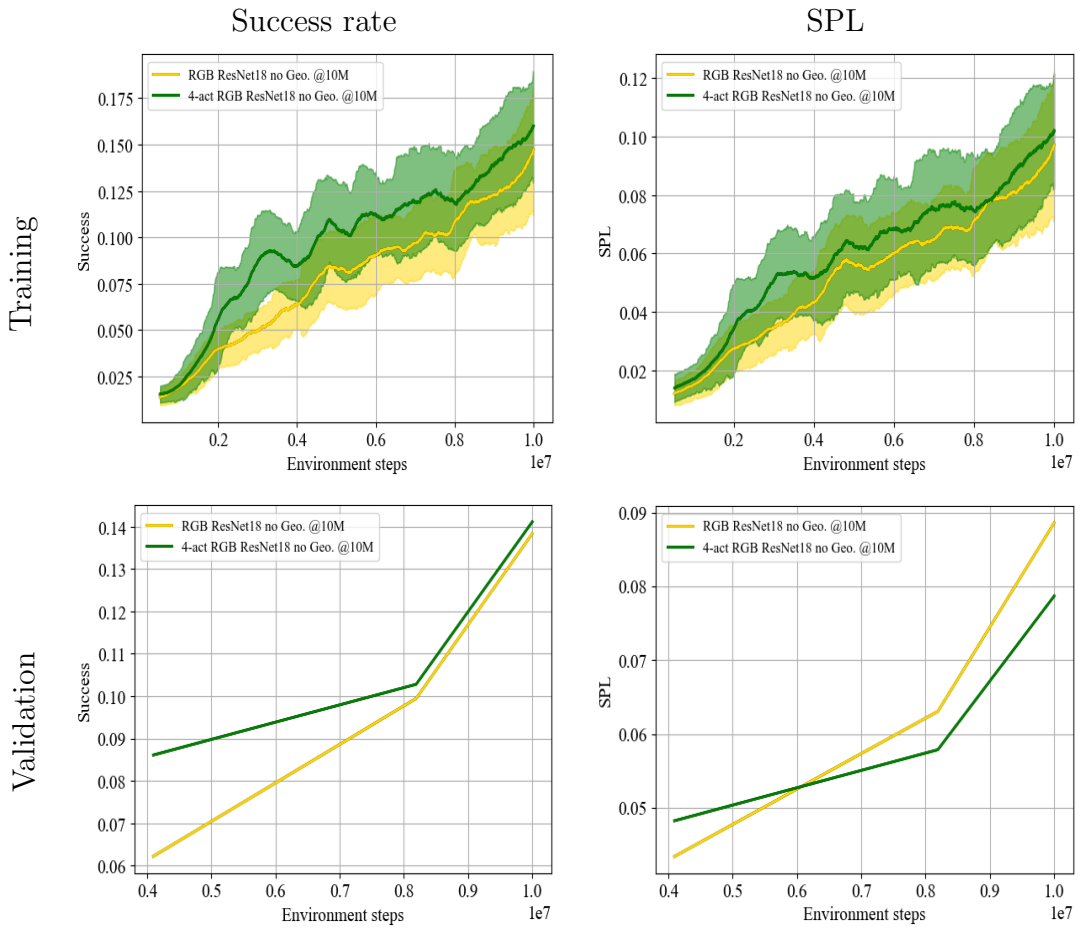

Figure 3.11: Training and validation curves of 6 and 4-action RGB ResNet18 agents trained without geodesic rewards for 10M environment interactions.

Table 3.7 reports the 4-action agent's performance across episode difficulties and object classes, and shows the overall success rate generalization error continues to be present (6.23%).

Table 3.7: Performance metrics of the 4-action RGB ResNet18 trained without geodesic rewards for 10M environment interactions.

Object	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.0294	0.0294	0.0067	0.0067
Apple	0.2778	0.1698	0.1200	0.0832
BaseballBat	0.1176	0.0891	0.0067	0.0033
BasketBall	0.4412	0.2427	0.2133	0.1151
Bowl	0.1395	0.0990	0.0533	0.0420
GarbageCan	0.2703	0.1400	0.2733	0.1652
HousePlant	0.4419	0.2280	0.4467	0.2213
Laptop	0.1463	0.0863	0.1000	0.0668
Mug	0.0698	0.0698	0.0733	0.0505
Television	0.3800	0.1582	0.3400	0.1375
SprayBottle	0.0556	0.0556	0.0400	0.0330
Vase	0.0244	0.0244	0.0200	0.0200
Difficulty	**Train success rate**	**Train SPL**	**Val success rate**	**Val SPL**
Easy	0.5421	0.2664	0.4121	0.2107
Medium	0.1508	0.1075	0.1080	0.0702
Hard	0.0482	0.0312	0.0232	0.0108
All episodes	0.2034	0.1167	0.1411	0.0787

Figures 3.12 and 3.13 compare the validation metrics of the 6-action baseline and RGB ResNet18 agents presented thus far (tables 3.4 to 3.6) against the 4-action RGB ResNet18 agent. While the 4-action RGB ResNet18 agent obtains a marginally improved validation success rate when compared to its 6-action counterpart, its overall SPL decreases, indicating lower trajectory efficiency. However, the validation performance metric differences between the 4 and 6-action RGB ResNet18 agents are relatively small and may be explained by the variance between runs. As such, reducing the action space does not demonstrate a clear improvement in either task completion or trajectory efficiency. Hence, the action space remains untouched for future experiments, and it is to be assumed that the full 6 actions available to the agent are used for the remainder of this thesis.

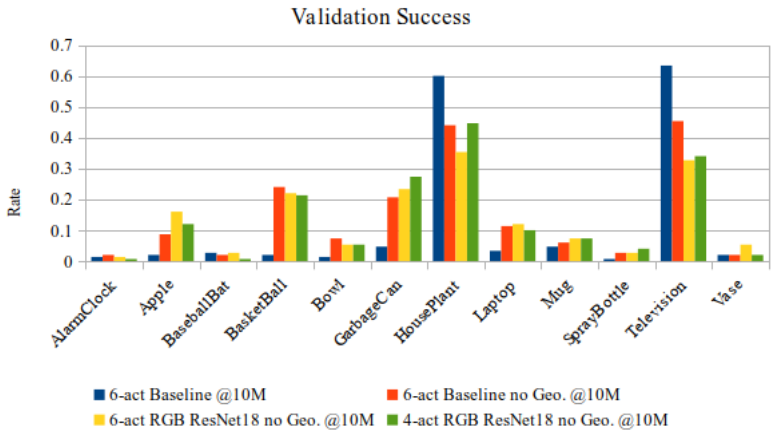

Figure 3.12: Object class validation success rate comparison between the 6-action agents (Baseline, Baseline no Geo., RGB ResNet18 no Geo.) and the 4-action RGB ResNet18 no Geo. agent.

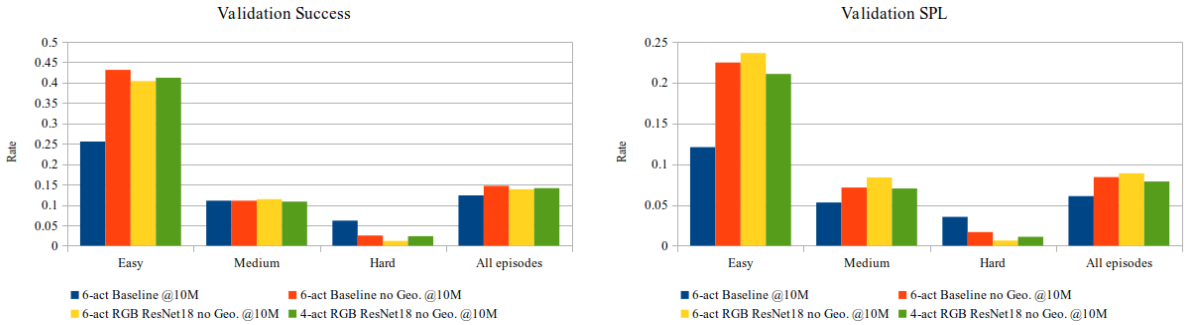

Figure 3.13: Episode difficulty validation metrics comparison between the 6-action agents (Baseline, Baseline no Geo., RGB ResNet18 no Geo.) and the 4-action RGB ResNet18 no Geo. agent.

3.3.4 Increasing Environment Interactions

Given the relative brevity of 10M environment interactions in comparison with the 170M of the RoboTHOR challenge baseline, the relationship between the number of environment

steps and performance is explored. Increasing environment interactions should improve training and validation metrics as well as highlight the influence of learning rate decay during optimization. The environment interactions are increased to 20M and 50M to investigate the influence of learning rate decay on training dynamics, while constraining wall clock time.

As expected, figure 3.14 shows a clear benefit to training the agent for longer periods of time. However, since the learning curves become flatter, it is also evident that the rate of improvement in both success and SPL is greatly reduced when increasing training time. This is likely influenced by a decreased slope in learning rate decay, coupled with the Adam optimizer [44] which uses a momentum component in its parameter updates. A range of research has investigated learning rate schedules when optimizing neural networks. Andrychowicz et al. [80] indicate that learning rate decay improved performance in four of five RL tasks, although the benefit was minimal. Smith [85] demonstrates that learning rate decay as well as cyclical learning rates can help in supervised learning. Gulde et al. [86] show improved performance with PPO using both triangular and exponentially decreasing cyclical learning rates. While such a detailed study is beyond the scope of this thesis, the experiments demonstrated in this section suggest that research into learning rate schedules may provide further insights in their application to RL tasks, including ObjectNav.

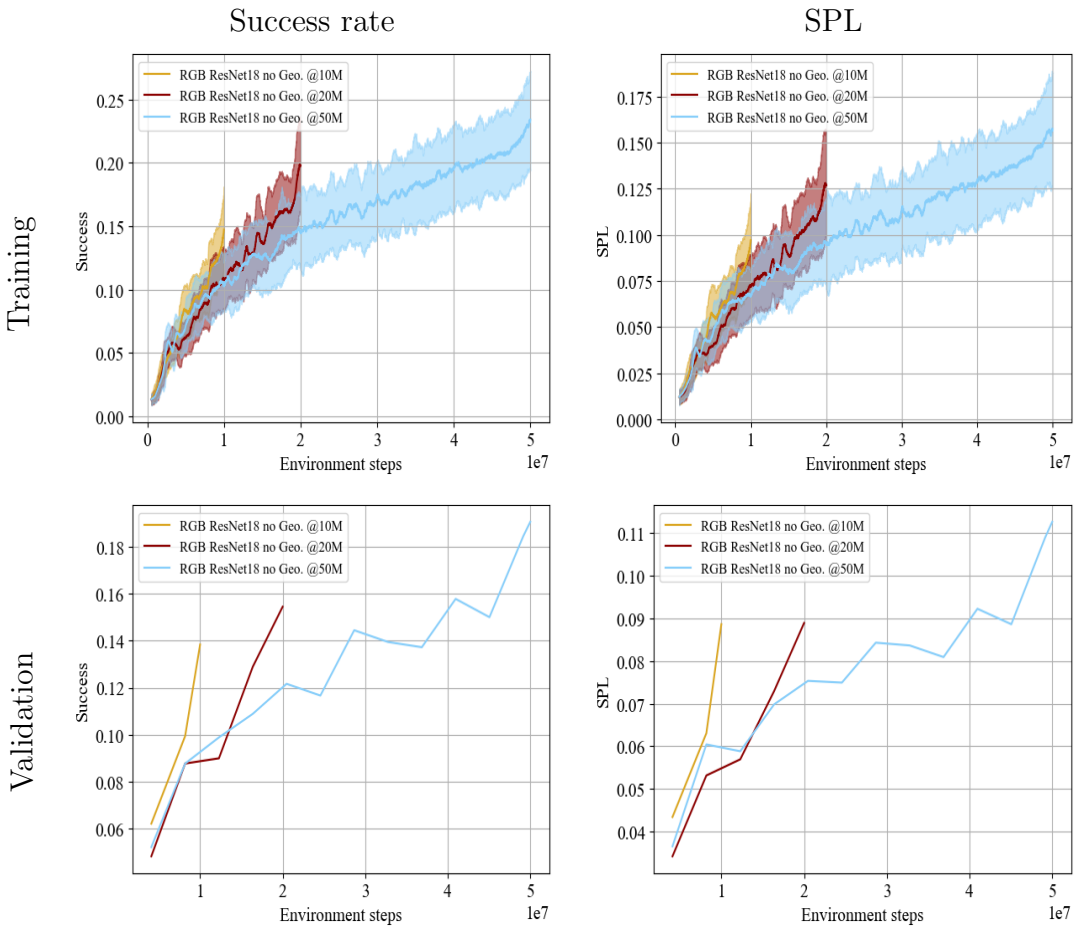

Figure 3.14: Training and validation curves of RGB ResNet18 agents trained without geodesic rewards and for 10M, 20M, and 50M environment interactions.

Table 3.8 compares performance metrics between agents trained for 10, 20, and 50 million steps. While increasing training time demonstrates improved overall training and validation metrics, it also shows that the generalization gap increases with training interactions (5.85% @10M, 9.86% @20M, 11.33% @50M). This suggests that the agent is increasingly overfitting to scenarios in the training dataset as more environment interac-

tions are afforded to it.

Table 3.8: Performance metrics of RGB ResNet18 agents trained for 10M, 20M, and 50M steps without the geodesic reward.

Object	RGB ResNet18 no Geo. @10M		RGB ResNet18 no Geo. @20M		RGB ResNet18 no Geo. @50M	
	Success rate	SPL	Success rate	SPL	Success rate	SPL
	Train | Val	Train | Val	Train | Val	Train | Val	Train | Val	Train | Val
AlarmClock	0.0851 | 0.0133	0.0697 | 0.0067	0.0882 | 0.0267	0.0805 | 0.0100	0.0513 | 0.0267	0.0512 | 0.0229
Apple	0.1818 | 0.1600	0.1458 | 0.1144	0.2439 | 0.1400	0.1538 | 0.0989	0.2830 | 0.1467	0.1849 | 0.1073
BaseballBat	0.0455 | 0.0267	0.0324 | 0.0241	0.0625 | 0.0067	0.0104 | 0.0000	0.0000 | 0.0333	0.0000 | 0.0332
BasketBall	0.3000 | 0.2200	0.2020 | 0.1070	0.6889 | 0.4333	0.4111 | 0.2150	0.5000 | 0.2533	0.3662 | 0.1407
Bowl	0.1053 | 0.0533	0.0849 | 0.0515	0.0455 | 0.0667	0.0409 | 0.0519	0.1667 | 0.0600	0.1627 | 0.0532
GarbageCan	0.2895 | 0.2333	0.1843 | 0.1582	0.3696 | 0.2933	0.2432 | 0.1752	0.4909 | 0.3133	0.2676 | 0.1991
HousePlant	0.5111 | 0.3533	0.3542 | 0.1906	0.3611 | 0.2867	0.1874 | 0.1468	0.6200 | 0.5600	0.3657 | 0.2684
Laptop	0.0811 | 0.1200	0.0590 | 0.1129	0.1591 | 0.1133	0.1124 | 0.0902	0.2059 | 0.1467	0.1406 | 0.1278
Mug	0.0286 | 0.0733	0.0286 | 0.0633	0.1250 | 0.0667	0.1046 | 0.0599	0.1429 | 0.0867	0.1141 | 0.0742
SprayBottle	0.0909 | 0.0267	0.0859 | 0.0221	0.0333 | 0.0867	0.0000 | 0.0562	0.0238 | 0.0400	0.0238 | 0.0399
Television	0.4706 | 0.3267	0.3098 | 0.1698	0.3684 | 0.3133	0.1934 | 0.1431	0.7544 | 0.5933	0.4240 | 0.2628
Vase	0.0652 | 0.0533	0.0652 | 0.0433	0.3182 | 0.0200	0.2293 | 0.0199	0.0500 | 0.0267	0.0417 | 0.0225
Difficulty	Train | Val	Train | Val	Train | Val	Train | Val	Train | Val	Train | Val
Easy	0.5000 | 0.4038	0.3181 | 0.2363	0.5568 | 0.4121	0.2892 | 0.2154	0.6142 | 0.4615	0.3070 | 0.2709
Medium	0.2105 | 0.1140	0.1662 | 0.0837	0.2849 | 0.1357	0.2030 | 0.0875	0.2952 | 0.1657	0.2224 | 0.1045
Hard	0.0262 | 0.0116	0.0224 | 0.0063	0.0397 | 0.0249	0.0277 | 0.0145	0.1214 | 0.0614	0.0968 | 0.0285
All episodes	0.1968 | 0.1383	0.1409 | 0.0887	0.2530 | 0.1544	0.1571 | 0.0889	0.3039 | 0.1906	0.1945 | 0.1127

Figures 3.15 and 3.16 compare the validation metric differences between the agents trained for 10, 20 and 50 million environment interactions. These figures show that performance metrics across object class and episode difficulty generally increase as environment interactions are increased. However, there are some target classes that perform better with less training interactions (e.g, BasketBall @20M > BasketBall @50M). These differences suggest that performance differences between one object and another are difficult to attribute to a single cause, although the number of objects that are learnt past a certain threshold can still show an agent's ability to learn multiple objects.

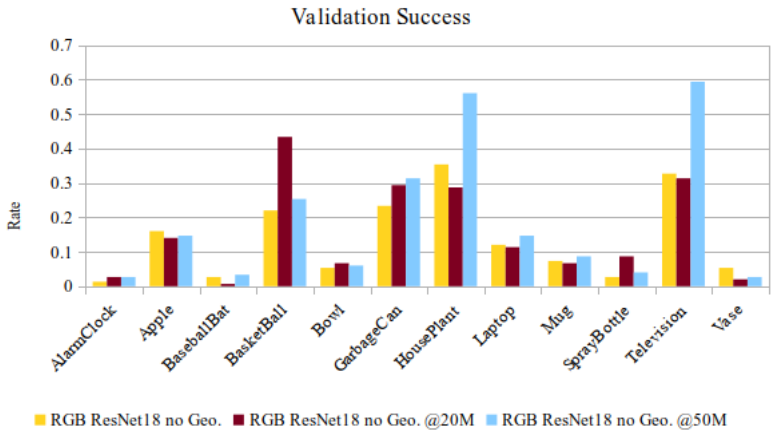

Figure 3.15: Object class validation success rate comparison of RGB ResNet18 agents trained for 10, 20, and 50 million environment interactions without geodesic rewards.

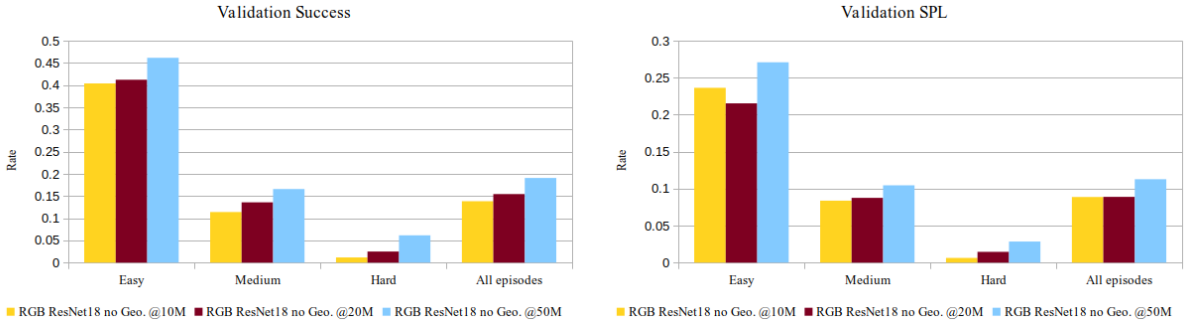

Figure 3.16: Episode difficulty validation metrics comparison of RGB ResNet18 agents trained for 10, 20, and 50 million environment interactions without geodesic rewards.

The training time for the 10M, 20M and 50M trained agents are roughly 21h, 37h and 86h respectively, indicating that the tradeoff between performance and training time for this RL task is not linear. The baseline's leaderboard benchmark for the RoboTHOR 2021 challenge was trained for 170M environment interactions, which would require unreasonably long training times per agent run in this thesis due to using PPO instead of DD-PPO (due

to a lack of GPUs). However; it is worth noting that the agent trained for 50M environment interactions obtains over half the validation success rate of the leaderboard baseline (19.06% vs 35.11%) with less than a third of the training experience. Although more benefits may be achieved by extending training, the experiments presented for the remainder of this thesis are capped to 50M environment interactions to constrain the wall clock time per experiment.

3.3.5 Advantage Normalization

Andrychowicz et al. [80] highlight a minimal difference between agents trained with and without advantage normalization. Advantage normalization is often applied per mini-batch and is a common implementation detail in multiple reinforcement learning libraries such as AllenAct [76] and Stable-Baselines3 [87]. Since there may be a limited benefit to applying such a normalization technique, its removal from the PPO algorithm is experimented with.

Figure 3.17 illustrates the training dynamics and validation curves of RGB ResNet18 agents trained with and without advantage normalization and without geodesic rewards for 50M environment interactions. Both the training and validation curves demonstrate an increasing trend throughout training without much difference between the two. The differences in validation performance are small enough that they could be due to the variance between runs.

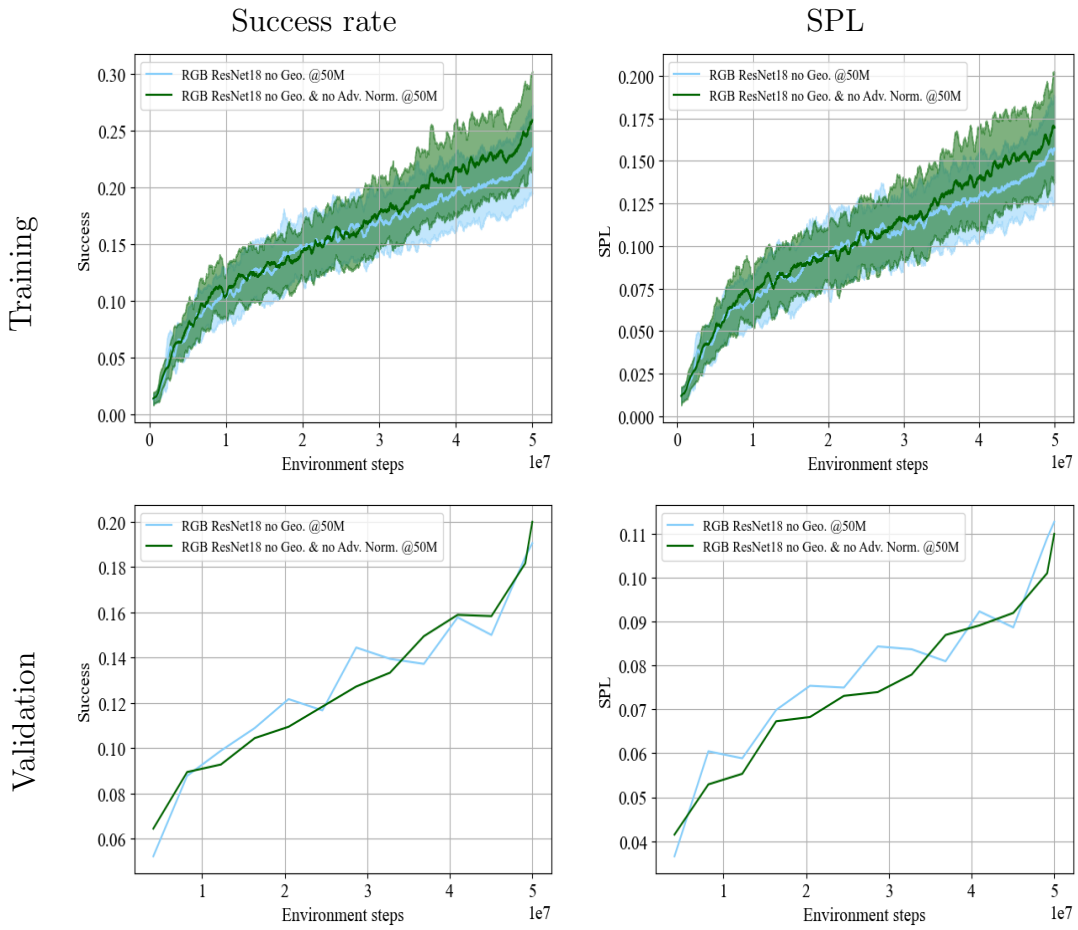

Figure 3.17: Training and validation curves of RGB ResNet18 agents trained with and without advantage normalization for 50M environment interactions without geodesic rewards.

Table 3.9 shows the performance metrics of the agent trained without advantage normalization across object classes and episode difficulties. It shows an overall success generalization gap of 12.6%, which is close to the 11.3% obtained by its counterpart that was trained with advantage normalization for the same amount of training interactions (@50M from table 3.8).

Table 3.9: Performance metrics of the RGB ResNet18 agent trained without the geodesic reward or advantage normalization for 50M environment interactions.

Object	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.0690	0.0643	0.0267	0.0200
Apple	0.2647	0.1955	0.1600	0.1059
BaseballBat	0.0714	0.0714	0.0467	0.0386
BasketBall	0.7429	0.4758	0.3933	0.1649
Bowl	0.2069	0.1751	0.0467	0.0364
GarbageCan	0.4444	0.2970	0.3333	0.2021
HousePlant	0.7619	0.4895	0.5267	0.2635
Laptop	0.0857	0.0578	0.1400	0.1056
Mug	0.1389	0.1299	0.0733	0.0636
SprayBottle	0.1282	0.1078	0.0533	0.0446
Television	0.8387	0.4889	0.5733	0.2475
Vase	0.0541	0.0541	0.0267	0.0265
Difficulty	**Train success rate**	**Train SPL**	**Val success rate**	**Val SPL**
Easy	0.5952	0.3647	0.4698	0.2629
Medium	0.3369	0.2456	0.1801	0.1007
Hard	0.1500	0.1071	0.0647	0.0303
All episodes	0.3260	0.2227	0.2000	0.1099

Figures 3.18 and 3.19 compare the RGB ResNet18 agent trained with and without advantage normalization and without geodesic rewards for 50M environment interactions. It shows that improvements with the BasketBall class accounts for most of the differences when omitting the normalization technique. However; as mentioned in section 3.3.4, comparisons between degrees of success on specific object classes remains difficult to attribute to a single cause. Figure 3.19 shows that validation performances across episode difficulty are also limitedly impacted by the use of advantage normalization. Both agents trained with and without advantage normalization achieve over half the overall validation success rate with less than a third of the training experience when compared to the AllenAct ResNet18 baseline's benchmark listed on the RoboTHOR 2021 challenge leaderboard (validation success rate of 35.11% @170M environment interactions). Given the limited impact of using advantage normalization when training the RGB ResNet18 agent, it is omitted

when training the agents presented in chapter 4.

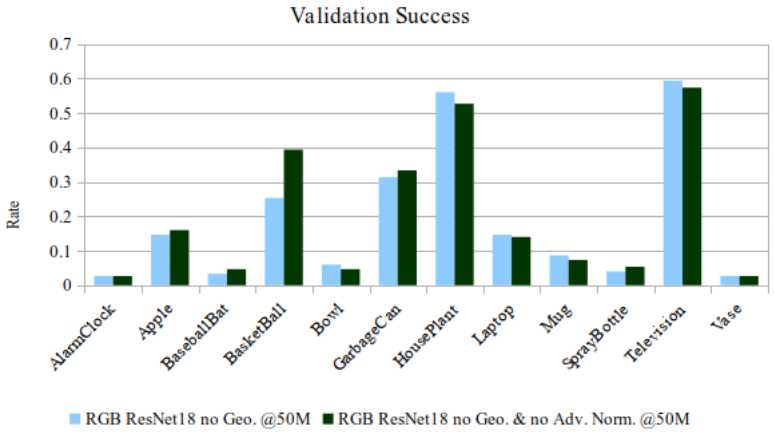

Figure 3.18: Object class validation success rate comparison of RGB ResNet18 agents trained with and without advantage normalization and without geodesic rewards.

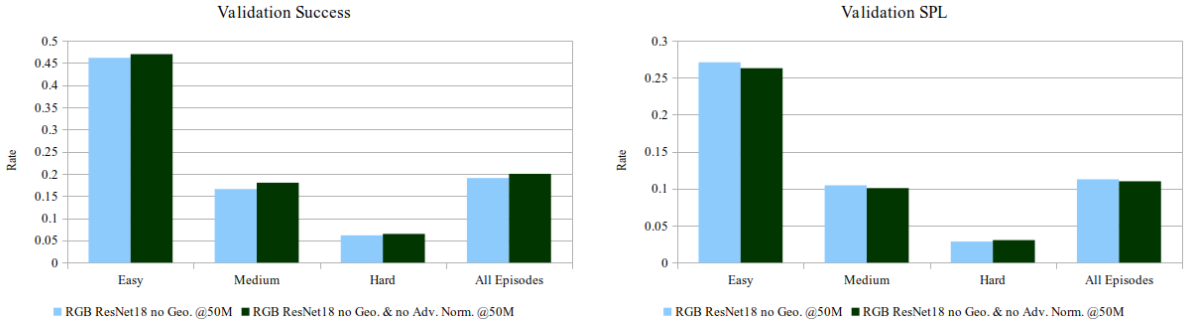

Figure 3.19: Episode difficulty validation metrics comparison of RGB ResNet18 agents trained with and without advantage normalization and without geodesic rewards.

3.3.6 Ablation Summary and Proposed Agent

Sections 3.3.1 to 3.3.5 report on an ablation study of different components of the AllenAct ResNet18 baseline. The reward scheme, observation space, action space, environment

interactions, and RL normalization technique were investigated to gain insight on how they effect ObjectNav success rate and SPL overall, as well as across object classes and episode difficulty. The findings can be summarized as:

- **Geodesic reward removal**
  - increases validation success rate in easy and medium episodes but decreases success in hard scenarios;
  - reduces validation metrics in Television and HousePlant object classes, but improves those metrics in GarbageCan and BasketBall classes;
  - improves overall validation success rates and SPL.
- **Depth observations stream removal**
  - slightly reduces the overall validation success rate while minimally improving validation SPL;
  - reduces training time by 40% by reducing the processing requirements on the hardware used to train the agent.
- **Action space reduction**
  - improves training curves but marginally impacts validation metrics, which increases the generalization gap;
  - negligibly increases validation success rate and reduces validation SPL.
- **Increasing environment interactions**

  - improves performance metrics when increased from 10 to 50 million interactions, but flattens learning curves, likely due to interactions with the learning decay hyperparameter and Adam optimization technique.

- **Advantage normalization**

  - demonstrates minimal impact on the ObjectNav agent's learning.

All agents presented in the ablation study demonstrate significant levels of generalization error in the objects that they learn, indicating that this still remains an issue. This is despite some agents showing improved validation metrics overall.

The RGB ResNet18 agent presented in section 3.3.5 is trained without geodesic rewards and without advantage normalization, and considers only RGB input. It is a culmination of the ablation study performed on the baseline AllenAct ResNet18 agent and is proposed as a leaner ObjectNav agent that obtains increased overall validation performance metrics. These improvements are achieved by removing dense geodesic rewards from the reward schedule, and removing the depth stream from the agent's model architecture but preserving a 6-action space and a compromised training over 50M environment interactions. The agent's reward schedule is described by

$$r = r_{slack} + Sr_{success} \tag{3.8}$$

where $r_{slack} = -0.01$, $r_{success} = 10$, and $S \in \{0, 1\}$ indicates success.

Its architecture is illustrated in figure 3.7, which only relies on RGB and target object observations. The model first processes RGB observations with a fixed ResNet18 net-

work [26] trained on ImageNet [24]. The resulting 512x7x7 tensor, I, is then compressed with a two layer convolutional neural network (CNN), with kernel sizes of 128x1x1 and 32x1x1 respectively, and ReLU activations, forming the 32x7x7 tensor, I'. A target object embedding is formed via an embedding layer and is then tiled to form G, a 32x7x7 goal tensor. This tensor is concatenated with I', and fed to another compression CNN. The resulting 32x7x7 tensor is flattened and fed to a 1-layer, 512 dimensional gated recurrent unit (GRU). The recurrent output is fed to linear layers $\pi_\theta(a_t|s_t)$ (the policy) and $V_\phi(s_t)$ (the state value function), where θ and ϕ represent the parameters of the respective linear layers. This architecture resembles the EmbCLIP agent [34], but differs in preprocessing network.

The proposed agent maintains the default action space: *move-forward 0.25m, turn-left 30°, turn-right 30°, look-up 30°, look-down 30°, stop.* This agent is constrained to 50M environment interactions to limit training time, although training and validation curves presented in section 3.3.4 show an increasing trend, suggesting that further training could improve overall performance metrics. The proposed agent is trained using PPO without advantage normalization, as the inclusion of this normalization technique demonstrates no benefit. The PPO hyperparameters used are the same as in table 3.3, except that max interactions are set to 50M, and w_{geo} is omitted as the geodesic reward is removed. Table table 3.10 summarizes the differences between the baseline and proposed agents.

Table 3.10: Differences between the baseline agent and proposed agent.

Agent	Reward scheme	Observation streams	Advantage normalization
Baseline	$r = r_{slack} + Sr_{success} - w_{geo}\Delta_{geo}$	RGB-D ResNet18 (figure 2.3)	Used
RGB ResNet18	$r = r_{slack} + Sr_{success}$	RGB ResNet18 (figure 3.7)	Not used

This agent achieves over half the overall validation success rate with less than a third of the training experience when compared to the baseline RGB-D ResNet18 benchmark listed on the RoboTHOR 2021 challenge leaderboard. However; it does have one problem that is amplified by the modifications made. Namely, it is less successful in sparse reward settings (i.e. hard episodes) than the baseline agent due to the removal of the dense geodesic reward, as demonstrated in figure 3.6. Chapter 4 studies this agent further and attempts to address the sparsity of rewards by incorporating exploration bonuses.

3.3.7 Comparison of Proposed Agent and Baseline

Given that the baseline agent presented in section 3.2 was only trained for 10M environment interactions, there is still a need to extend this baseline's training to allow for fairer comparisons to the agent proposed in section 3.3.6. Hence, the baseline agent is trained for 50M environment interactions to enable fairer comparisons to the RGB ResNet18 agent trained without geodesic rewards and without advantage normalization. This helps ensure that performance improvements are not solely attributable to increased environment interactions.

Figure 3.20 shows the training and validation curves of performance metrics for both the baseline agent and the RGB ResNet18 agent trained without geodesic rewards and without advantage normalization. It demonstrates that the original baseline plateaus roughly after 25M environment interactions while the proposed agent continues an increasing trend in both success rate and SPL throughout training. The figure also shows that the proposed agent's overall validation metrics are significantly greater than the original baseline.

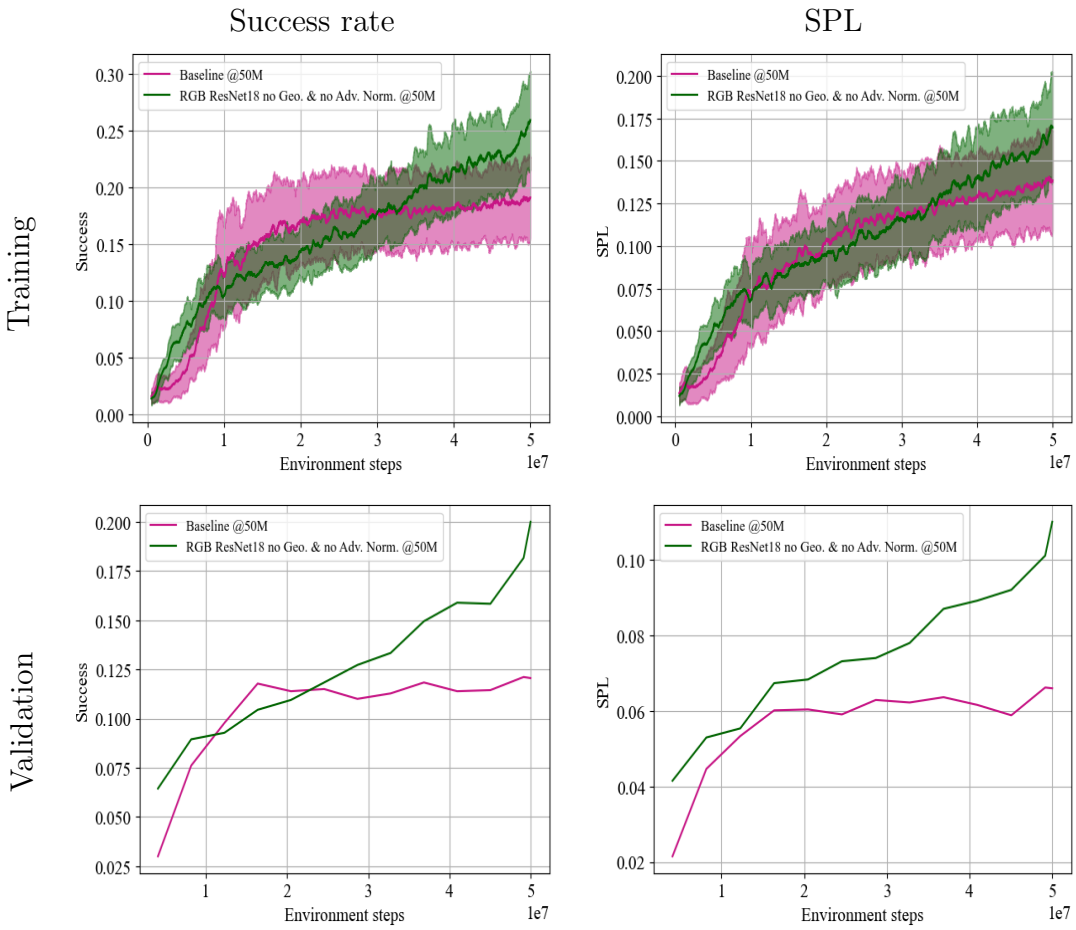

Figure 3.20: Training and validation curves of the baseline agent and the RGB ResNet18 agent trained without geodesic rewards and without advantage normalization. Each agent is trained for 50M environment interactions.

Table 3.11 shows the baseline agent's performance metrics across object class and episode difficulty. The baseline continues to demonstrate a large generalization gap for HousePlant and Television classes (28.5% and 41.2% respectively) despite increasing training. It also shows that the baseline agent mostly learns those two object classes and ignores others.

Table 3.11: Performance metrics of the baseline agent trained for 50M environment interactions.

	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.0270	0.0270	0.0067	0.0067
Apple	0.0769	0.0769	0.0267	0.0258
BaseballBat	0.0000	0.0000	0.0200	0.0200
BasketBall	0.0556	0.0334	0.0333	0.0267
Bowl	0.0625	0.0625	0.0133	0.0133
GarbageCan	0.0790	0.0734	0.0467	0.0465
HousePlant	0.9583	0.6993	0.6733	0.3406
Laptop	0.0526	0.0480	0.0200	0.0200
Mug	0.0000	0.0000	0.0267	0.0267
SprayBottle	0.1333	0.1198	0.0133	0.0071
Television	0.9787	0.6827	0.5667	0.2537
Vase	0.0313	0.0313	0.0067	0.0065
Difficulty	Train success rate	Train SPL	Val success rate	Val SPL
Easy	0.3053	0.2236	0.2390	0.1212
Medium	0.2700	0.1920	0.1032	0.0574
Hard	0.1648	0.1376	0.0746	0.0449
All episodes	0.2378	0.1781	0.1211	0.0661

Figures 3.21 and 3.22 compare the baseline agent's validation metrics against those of the proposed RGB ResNet18 agent trained without geodesic rewards and without advantage normalization. Figure 3.21 demonstrates that the baseline agent obtains greater validation success in the HousePlant class while being less successful in all other object classes when compared to the RGB ResNet18 agent. Figure 3.22 shows that the baseline agent is more successful in hard episode difficulties, but is worse in easy and medium difficulty episodes. It should be noted that both agents still struggle with sparse reward episodes (i.e. hard episodes).

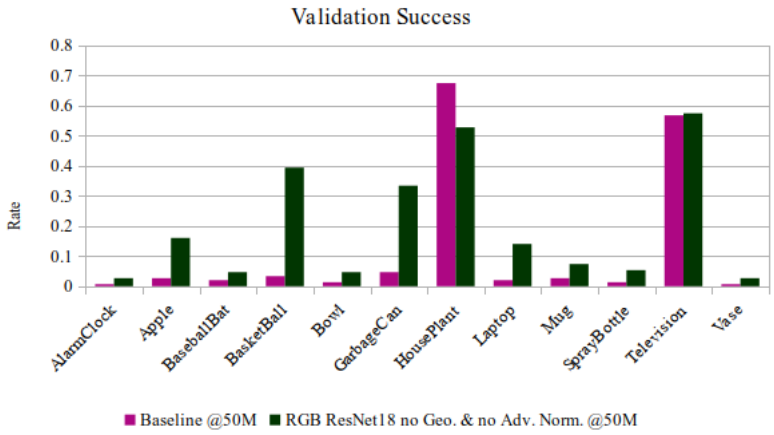

Figure 3.21: Object class validation success rate comparison between the baseline agent and the RGB ResNet18 agent trained without geodesic rewards or advantage normalization.

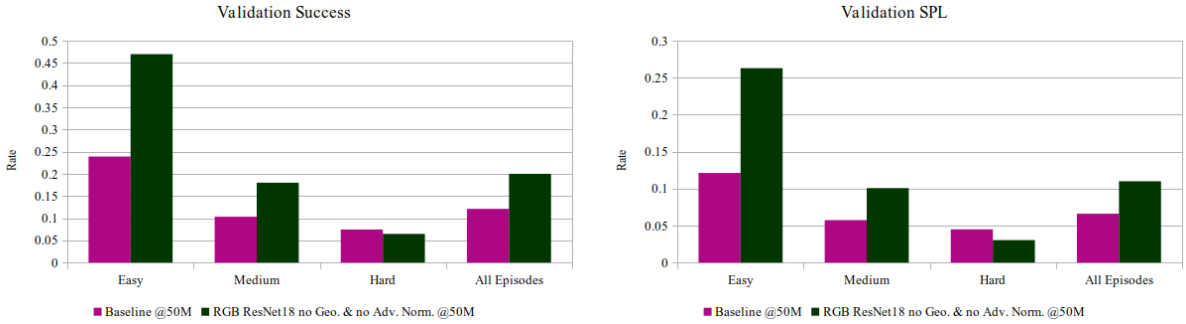

Figure 3.22: Episode difficulty validation metrics comparison between the baseline agent and the RGB ResNet18 agent trained without geodesic rewards and without advantage normalization.

Figure 3.23 demonstrates two sample trajectories of the baseline and RGB ResNet18 agent from the validation set of episodes. We can see that both agents are capable of exploring to locate the Television, but demonstrate short, unsuccessful paths for the Apple class. This is likely due to the agents learning the Television class well but not learning

to locate the Apple class. Additionally, the baseline agent has a much more direct path when searching for the television, while the proposed agent is less direct. This is likely a consequence of the geodesic reward being omitted from the RGB ResNet18 agent. motivates the investigation of exploration bonuses as an alternative to the geodesic reward to be conducted in chapter 4.

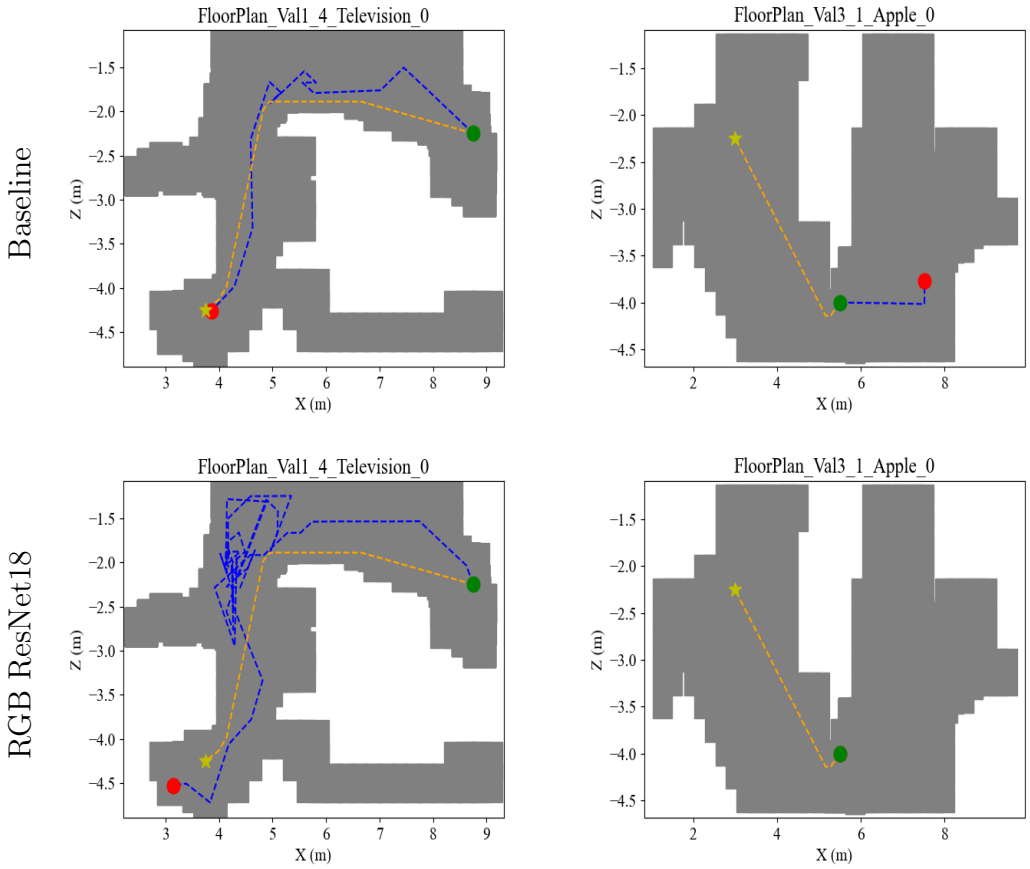

Figure 3.23: Television (left) and Apple (right) sample trajectories of baseline and proposed agents. Green and red dots indicate starting and end points respectively, stars indicate target location, blue lines indicate agent trajectories, and orange lines indicate the shortest path. Note that the red dot is hidden under the green dot for the RGB ResNet18 sample Apple trajectory.

While the training curves in figure 3.20 suggest the RGB ResNet18 agent will continue to improve in performance and that the baseline agent will not learn further, it is expected that the baseline agent reaches the leaderboard benchmark (i.e., 35% validation success rate) when trained for 170M environment interactions. Hence, it is not guaranteed that the RGB ResNet18 will outperform the baseline with more training interactions. However, it is reasonable to predict that the findings of the ablation study hold for greater environment interactions. Specifically:

- the geodesic reward signal, as demonstrated in section 3.3.1 and by Maksymets et al. [21], reinforces direct routes to target objects, likely overfitting an agent's policy to the training episodes;
- depth observations, as demonstrated in section 3.3.2 and shown by the EmbCLIP agent [34], are likely not crucial to the baseline;
- the full action space is expected to remain necessary since observing target objects located at different heights is an important part of the ObjectNav problem;
- and advantage normalization, as demonstrated in section 3.3.5 and supported by Andrychowicz et al. [80], is expected to not be crucial for agent learning with PPO.

3.4 Summary

This chapter detailed the RoboTHOR 2021 ObjectNav challenge dataset and the baseline AllenAct ResNet18 agent while demonstrating the challenges of generalization, feature representation, and reward sparsity. Section 3.3 described an ablation study on the AllenAct

ResNet18 baseline and proposed modifications to it with the objective of improving generalization while considering the impacts on training time. Specifically, it was shown that removing the geodesic reward improves validation performance within 10M steps, and that depth sensing provided limited benefits at the cost of a 67% increase in training time. Extending the span of training from 10M to 20M and 50M environment interactions improves the overall validation performance, as one might expect, but the learning curves become flatter such that it takes longer to obtain similar performance metrics. It is conjectured that this is likely due to the interplay between the annealed learning rate and the Adam optimizer. Removal of vertical camera rotations shows negligible improvements in success rate but lowers SPL, while PPO with advantage normalization does not perform much differently when the normalization technique is omitted for this ObjectNav task. Finally, the original baseline is run for 50M environment interactions to evaluate the modifications made to the baseline. It is shown that the proposed RGB ResNet18 agent trained without geodesic rewards and without advantage normalization performs much better in both success rate and SPL in validation episodes. These results lead to the adoption of the RGB ResNet18 agent that uses the full 6-action space, and is trained without a geodesic dense reward and without advantage normalization, for 50M environment interactions, which is further studied in Chapter 4.

Chapter 4: Object Goal Navigation Using Exploration Bonuses

The previous chapters sought to identify challenges related to ObjectNav and to propose a RGB ResNet18 agent that improves validation success rate and success weighted by path length (SPL) while balancing training time and performance. Still, this agent shows poor performance as episode difficulty, determined by the shortest path length to the goal, increases. This illustrates the difficulty of training reinforcement learning (RL) agents in sparse reward settings. The original baseline [76] partially deals with this issue by utilizing the reduced geodesic reward bonus, but it comes at the expense of poor generalization, and poor success with objects other than the Television and HousePlant object classes.

This chapter will seek to mitigate the effects of sparse rewards in the ObjectNav task by examining exploration intrinsic rewards. Section 4.1 investigates episodic exploration bonuses and proposes two adaptations of a simplified "ExploreTillSeen" [21] reward scheme to train the RGB ResNet18 agent proposed in section 3.3.6. Section 4.2 describes the predictive error bonuses applied to that same RGB ResNet18 agent via random network distillation (RND) [16] and presents an original "RNDTillSeen" reward scheme. Section 4.3 presents an original "NGUTillSeen" ObjectNav agent trained by combining long-term novelty with episodic curiosity.

4.1 Episodic Exploration

The Red-Rabbit agent [20] attempts to deal with sparse rewards by using auxiliary tasks and count-based exploration bonuses with a tethered policy. The count-based reward is

computed by splitting the scene into a two-dimensional grid, and counting the agent's occupancy in its current cell. The resulting reward scheme used by the Red-Rabbit agent is

$$r = r_{success} + r_{slack} + r_{explore} \tag{4.1}$$

$$r_{success} = 2.5 \text{ upon success} \tag{4.2}$$

$$r_{slack} = -10^{-4} \tag{4.3}$$

$$r_{explore} = 0.25\frac{d^t}{v} \tag{4.4}$$

where d is the decay rate of the exploration term, v is the count of the currently occupied cell, and t is the current episode step.

Ye et al. [20] further attempt to use the intrinsic curiosity module (ICM) [17] to reward exploratory behavior but could not yield improved results. Maksymets et al. [21] also use an episodic exploration bonus in their ExploreTillSeen reward scheme, which is computed as the change in the percentage of area explored by the agent. The agent's viewing frustum is projected onto a two-dimensional map of the scene to calculate the area explored. A "target seen" reward additionally reduces the sparsity of positive rewards, such that when the goal class mask occupies more than 3% of the field of view, the agent receives 30% of the reward for completing the task. The reward scheme is described by equation (2.58).

These approaches make use of episodic exploration bonuses to encourage exploratory policies by reducing the sparsity of rewards in the ObjectNav task. Given the improve-

ments enabled by these approaches, episodic rewards are explored by training the RGB ResNet18 agent presented in section 3.3 with two newly adapted variations of a simplified ExploreTillSeen reward scheme. Firstly, the manner in which a goal object is considered "seen" is simplified: the RoboTHOR simulation's flag for object visibility is used instead of the object's corresponding segmentation mask. This approach reduces processing per environment interaction and is useful given the agent's lack of a semantic segmentation modality. The proposed reward schemes differ simply by mode of exploration reward computation. The first uses a fixed auxiliary reward, r_{delta}, for moving to a new grid cell, which is computed as the inverse of total cells in a scene. This is a simplification of the original exploration bonus presented in [21] that reduces processing by eliminating the need to project the agent's viewing frustum onto a 2D plane. The second proposes a count-based computation for exploration bonuses, r_{count}. This computation is similar to [20], except for the absence of the decay term, and the inclusion of an exploration ratio that varies by the number of visitable cells in a scene. The combination of count-based rewards with ExploreTillSeen reward schedules has not been previously explored in an ObjectNav setting. The two exploration rewards become

$$r_{delta} = r_{expl} \frac{v_t - v_{t-1}}{n_{grid}} \tag{4.5}$$

$$r_{count} = \frac{r_{expl}}{N(x) n_{grid}} \tag{4.6}$$

where r_{expl} is the exploration weight, v_t is the number of visited cells at episode timestep, t, n_{grid} is the total number of scene grid cells (obtained by segmenting the reachable scene into a grid of 0.25m by 0.25m cells), and $N(x)$ is the visitation count of cell x, which is computed in a tabular manner. That is, the more environment steps an agent spends in a

cell, the smaller the exploration bonus becomes for being in that cell. The proposed delta and count-based ExploreTillSeen variants are formulated as

$$r = r_{slack} + \begin{cases} r_i & \text{until goal is seen} \\ r_{goal\_seen} & \text{first time goal is seen} \\ -\Delta_{geo} + S r_{success} & \text{otherwise} \end{cases} \tag{4.7}$$

where S is a binary flag indicating a successful stoppage by the robotic agent, Δ_{geo} is the geodesic reward and $r_i \in \{r_{delta}, r_{count}\}$. It is worth noting that the geodesic reward is only applied once the agent has "seen" the target object. Maksymets et al. [21] demonstrate that this reduces generalization error, which is also observed in section 3.3.1 when the geodesic reward is removed. Furthermore, the exploratory actions are not penalized until the target object is observed.

The configuration for these episodic reward agents follows the RGB ResNet18 agent proposed in section 3.3, except in varying the reward scheme to equation (4.7) with the coefficient values defined in table 4.1. The values for r_{expl} and r_{goal_seen} were initially adopted from the original ExploreTillSeen reward scheme, but the large r_{expl} value (25) biased the agent towards infinite exploration. This is likely due to the relative scale between exploration and the success reward. As such, experiments that followed reduced the r_{expl} term until such a pure exploration behavior was avoided. The reward scheme coefficients used in both delta and count ExploreTillSeen variants are shown in table 4.1.

Table 4.1: Delta and count ExploreTillSeen reward scheme coefficients.

Variable	Value
r_{expl}	5
r_{goal_seen}	3
$r_{success}$	10
r_{slack}	-0.01

Figure 4.1 illustrates the training dynamics and validation performance metrics of RGB ResNet18 agents trained using three reward schedules: equation (3.8), ExploreTillSeen-delta, and ExploreTillSeen-count. Both episodic bonus reward variants improve training and validation curves compared to when they are unused. Furthermore, both variants present an increasing trend that suggests more environment interactions could yield improved results. Ultimately, the agent trained with the count-based ExploreTillSeen scheme demonstrates better training and validation performances overall.

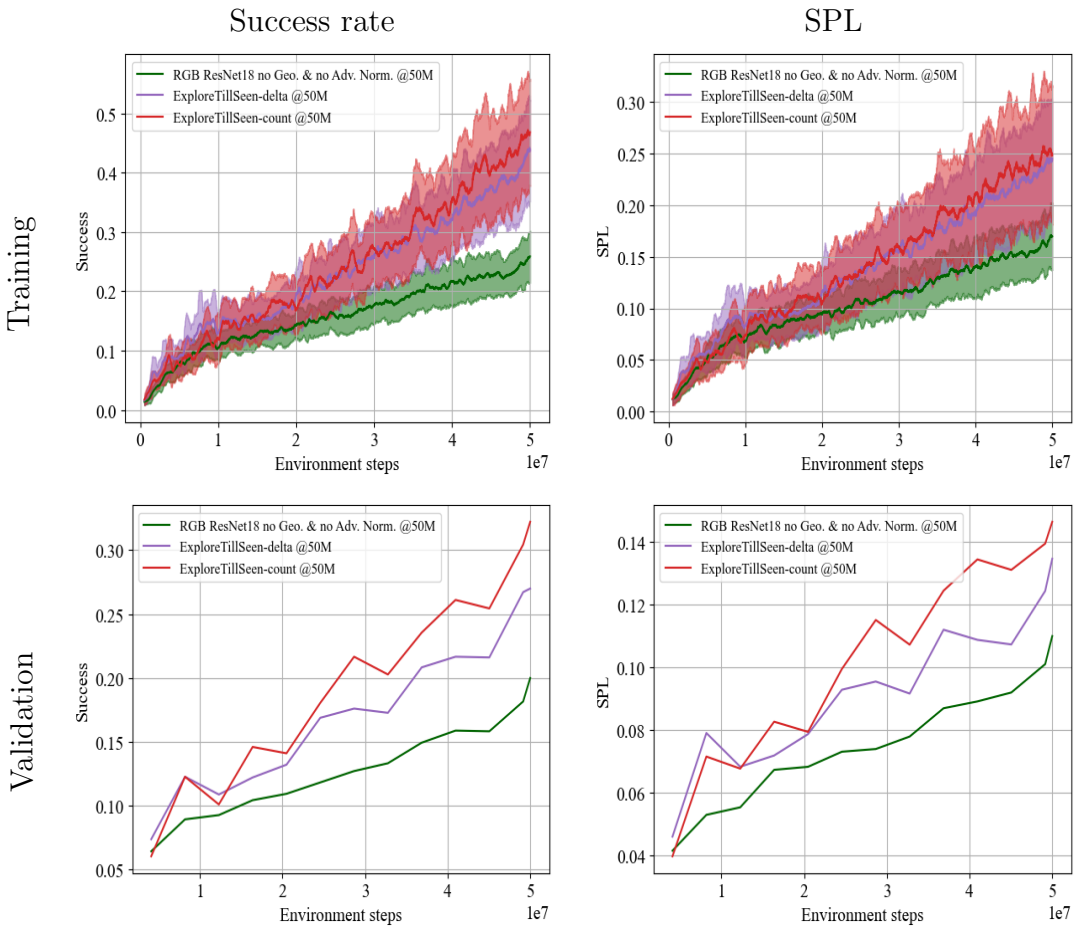

Figure 4.1: Training and validation curves of delta and count ExploreTillSeen agents compared against the RGB ResNet18 agent proposed in section 3.3.6.

Table 4.2 highlights the performance metrics of both delta and count ExploreTillSeen agents across object type and difficulty. It demonstrates that the delta and count-based reward schemes result in large overall success generalization gaps of 28% and 26.8%, proving generalization is still an issue for these agents.

Table 4.2: Performance metrics of the delta and count ExploreTillSeen agents' trained for 50M environment interactions.

Object	ExploreTillSeen-delta		ExploreTillSeen-count	
	Success rate	SPL	Success rate	SPL
	Train | Val	Train | Val	Train | Val	Train | Val
AlarmClock	0.3636 | 0.0800	0.2190 | 0.0465	0.0000 | 0.0933	0.0000 | 0.0582
Apple	0.8571 | 0.4200	0.3861 | 0.1993	0.7778 | 0.5133	0.5261 | 0.1954
BaseballBat	0.2222 | 0.0267	0.2222 | 0.0133	0.2222 | 0.0400	0.2222 | 0.0344
BasketBall	0.8667 | 0.4800	0.4109 | 0.2311	0.6154 | 0.5533	0.3290 | 0.2318
Bowl	0.3333 | 0.1067	0.1504 | 0.0445	0.0000 | 0.0800	0.0000 | 0.0630
GarbageCan	1.0000 | 0.6200	0.5306 | 0.3237	0.9231 | 0.7133	0.4888 | 0.3447
HousePlant	0.8182 | 0.5067	0.4288 | 0.2355	0.8000 | 0.5867	0.3405 | 0.2299
Laptop	0.6000 | 0.1933	0.3301 | 0.1220	0.8571 | 0.3267	0.5600 | 0.1532
Mug	0.2000 | 0.1200	0.1849 | 0.0930	0.2000 | 0.0933	0.1982 | 0.0734
SprayBottle	0.0714 | 0.1067	0.0714 | 0.0660	0.0000 | 0.0600	0.0000 | 0.0390
Television	0.9000 | 0.5600	0.4553 | 0.2204	0.8571 | 0.6867	0.5065 | 0.2894
Vase	0.0000 | 0.0200	0.0000 | 0.0200	0.4286 | 0.1200	0.3510 | 0.0444
Difficulty	Train | Val	Train | Val	Train | Val	Train | Val
Easy	0.7727 | 0.5055	0.3527 | 0.2414	0.8462 | 0.5467	0.5069 | 0.2365
Medium	0.6170 | 0.2569	0.3200 | 0.1316	0.6364 | 0.3037	0.3737 | 0.1464
Hard	0.3922 | 0.1459	0.2383 | 0.0743	0.5000 | 0.2123	0.3020 | 0.0921
All episodes	0.5500 | 0.2700	0.2913 | 0.1346	0.5900 | 0.3222	0.3523 | 0.1464

Table 4.3 summarizes the validation performance across episode difficulties for the baseline, RGB ResNet18 (from section 3.3.6), and delta and count ExploreTillSeen agents. Note that the "Baseline @170" metrics in table 4.3 come from the RoboTHOR 2021 leaderboard while the "Baseline @50M" refers to the baseline presented in section 3.3.7.

Table 4.3: Validation performance comparison between baseline, RGB ResNet18, and delta and count ExploreTillSeen agents.

Agent	Val success rate				Val SPL			
	Easy	Medium	Hard	Overall	Easy	Medium	Hard	Overall
ProcTHOR Fine-Tune [77]	—	—	—	0.6639	—	—	—	0.2744
EmbCLIP [34]	—	—	—	0.5222	—	—	—	0.2599
Baseline @170M	—	—	—	0.3511	—	—	—	0.1737
Baseline @50M	0.2390	0.1032	0.0746	0.1211	0.1212	0.0574	0.0449	0.0661
RGB ResNet18 no Geo. & no Adv.Norm @50M	0.4698	0.1801	0.0647	0.2000	0.2629	0.1007	0.0303	0.1099
ExploreTillSeen-delta @50M	0.5055	0.2569	0.1459	0.2700	0.2414	0.1316	0.0743	0.1346
ExploreTillSeen-count @50M	0.5467	0.3037	0.2123	0.3222	0.2365	0.1464	0.0921	0.1464

Table 4.3 shows that the delta trained variant vastly improves the overall validation success rates across all difficulties (+3.6 easy, +7.7% medium, +8.1% hard) when compared to the RGB ResNet18 agent trained without the exploration reward scheme. This agent shows an even greater performance gap over the baseline agent trained for 50M environment interactions. As intended, most improvements are in the medium and hard difficulty episodes, proving that the delta ExploreTillSeen reward scheme mitigates the challenge of sparse positive rewards. The count-based ExploreTillSeen agent shows even further improvements across difficulties when compared to the delta ExploreTillSeen scheme: +4.12% on easy, +4.7% on medium, and +6.64% on hard episodes. Hence, the count-based ExploreTillSeen agent demonstrates the best performance in all difficulties when compared to agents trained for the same number of environment interactions (50M).

When comparing the count-based ExploreTillSeen agent to the original AllenAct baseline agent listed on the RoboTHOR 2021 leaderboard, the validation success rate falls within 3%[1]. Thus, the count-based ExploreTillSeen agent obtains comparable results to the original baseline while experiencing less than a third (50M steps) of the environment interaction (170M in the original leaderboard baseline). The increasing training and validation curves in figure 4.1 suggest that this gap may be bridged or even surpassed if training were extended.

It is important to note that the EmbCLIP and ProcTHOR agents obtain even greater performances by addressing feature representation with the CLIP network. Additionally, ProcTHOR improves performance by addressing generalization error via pretraining on a larger dataset of ObjectNav scenes. These agents were published during the writing of this

[1]According to the RGB+D ResNet18-ImageNet submission's validation success rate from https://leaderboard.allenai.org/robothor_objectnav/submissions/public

thesis and they demonstrate that, aside from reward sparsity, generalization and feature representation are important problems to address in the ObjectNav task.

Figure 4.2 demonstrates two sample trajectories of the ExploreTillSeen-count agent in the same validation scenarios presented in figure 3.23. It shows the agent is capable of locating the Television and Apple while demonstrating exploratory behavior. These samples contrast figure 3.23, which shows the baseline and RGB ResNet18 agents exhibiting less exploration, especially for the Apple scenario.

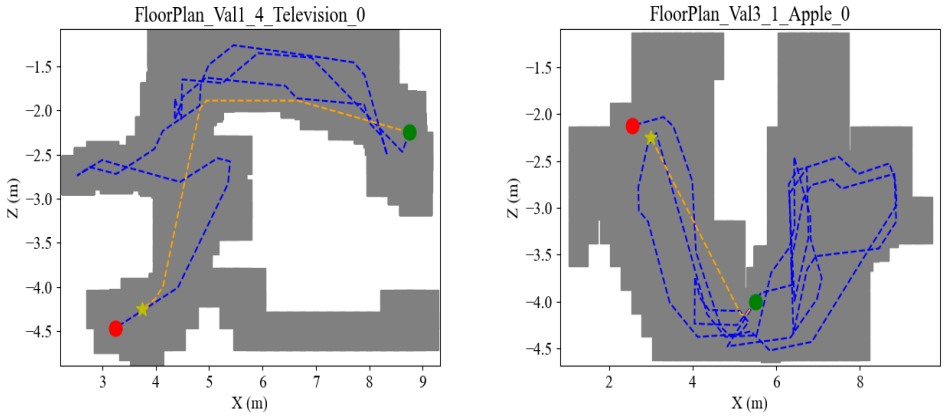

Figure 4.2: Television (left) and Apple (right) sample trajectories of the ExploreTillSeen-count agent. Green and red dots indicate starting and end points respectively, stars indicate target location, blue lines indicate agent trajectories, and orange lines indicate the shortest path.

4.2 Prediction Error Bonuses

As mentioned in section 2.2.4, there are many methods endowing RL agents with intrinsic curiosity. Burda et al. [59], for example, demonstrated successful policies using an intrinsic

curiosity module (ICM) in numerous games, especially in those with sparse rewards. This method computes exploration bonuses as the mean squared error (MSE) between predicted and observed states while simultaneously training a feature network using inverse dynamics. In the same vein, random network distillation (RND) computes exploration bonuses as the MSE between a predictor network and a randomly initialized and fixed oracle network [16]. RND was used at the time it was developed to achieve state-of-the-art performance on Montezuma's Revenge, a game with very sparse rewards.

These prediction error bonuses express long-term novelty such that observed states provide reduced intrinsic rewards as training progresses. Hence, to obtain larger intrinsic rewards, agents must seek novel states. Both ICM and RND approaches experiment with combining extrinsic and intrinsic reward streams, as well as only using the latter. They find that extrinsic rewards could be achieved despite their absence during training. In the RoboTHOR ObjectNav task, intrinsic-only agents would not be incentivized to find target objects. As such, this section explores incorporating the two rewards streams using prediction error bonuses via random network distillation, as it demonstrates better performance on sparse reward tasks in the arcade learning environment (ALE) [9]. Until the time of writing this thesis, prediction error intrinsic rewards, such as RND, had not been applied successfully to improve results in the ObjectNav tasks; although Ye et al. [20] remark that the ICM fails to improve their ObjectNav agent's performance. The following ObjectNav agents presented are another attempt to improve agent exploration in this task by using random network distillation.

The two proposed RND ObjectNav agents augment the RGB ResNet18 with RND. The first is referred to as the "RND" agent and the second with the newly adapted reward

scheme is labeled "RNDTillSeen". Instead of using the episodic bonus reward schemes presented in section 4.1, the RND agent receives a prediction error bonus for every environment interaction. Additionally, the RNDTillSeen reward scheme receives a RND prediction error bonus until the target object is "seen". Both agents use the RGB ResNet18 architecture (illustrated in figure 3.7 and the bottom portion of the network in figure 4.3), except that the ResNet18-processed RGB observations are fed to target and predictor networks, $\Phi(s_t)$ and $\hat{\Phi}(s_t)$, which are illustrated at the top of figure 4.3. Each of these networks contain CNNs composed of two convolutional layers joined by LeakyReLU activations. The target network then follows with one linear layer, whereas the predictor network follows with three that are joined by ReLU activations. The agent's overall architecture is depicted in figure 4.3 with layer widths and intermediate tensor dimensions. All CNN layer strides and paddings are set to one and zero respectively.

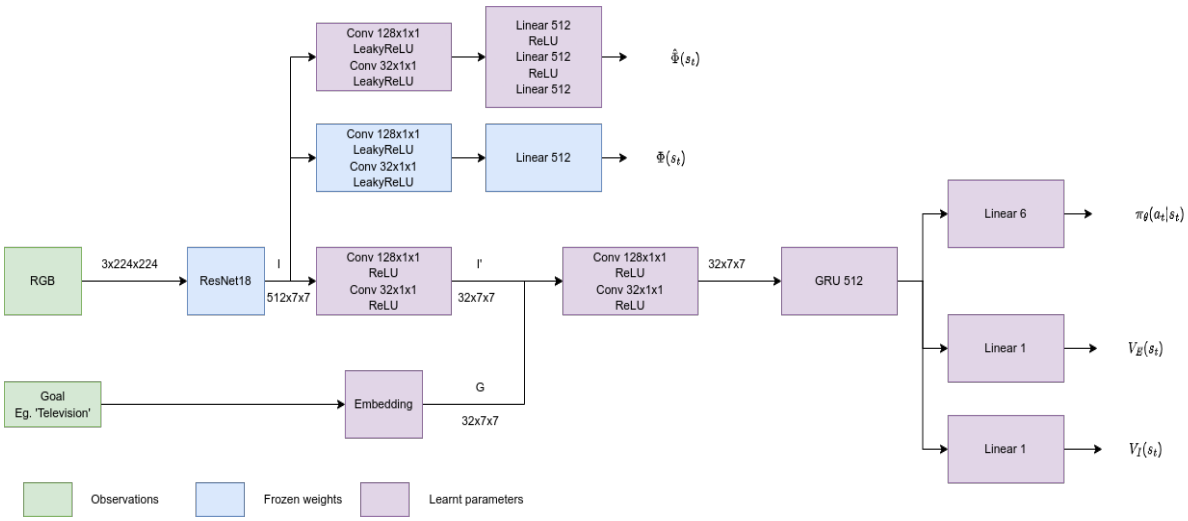

Figure 4.3: Proposed RND ObjectNav agent architecture overview.

Instead of adding intrinsic rewards to the total reward, the value function is split into

$V(s) = V_I(s) + V_E(s)$, and the proximal policy optimization (PPO) loss described by equation (3.6) is adapted to include the prediction MSE, as demonstrated in Burda et al. [16]. Equation (4.9) shows the inclusion of the prediction MSE (i.e. the $(\Phi(s_t) - \hat{\Phi}(s_t))^2$ term) in the PPO loss.

$$\mathcal{J}_\pi(\theta) = \min\left(\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})} A(s_t), clip\left(\frac{\pi(s_t, a_t, \theta)}{\pi(s_t, a_t, \theta_{old})}, 1-\epsilon, 1+\epsilon\right) A(s_t)\right) \tag{4.8}$$

$$\mathcal{J}(\theta, \theta_v) = \mathbb{E}\left[\mathcal{J}_\pi(\theta) - c_V(V(s_t) - \hat{V}(s_t, \theta_v))^2 - (\Phi(s_t) - \hat{\Phi}(s_t))^2 + c_e H\right] \tag{4.9}$$

Split value heads mean that generalized advantage estimation (GAE) now computes two advantages per reward stream and can apply different discount factors, γ_I, γ_E, per stream. The advantage function becomes $A(s_t) = \beta_I A_I(s_t) + \beta_E A_E(s_t)$, where β_I and β_E weight intrinsic and extrinsic rewards respectively, and $A_I(s_t)$ and $A_E(s_t)$ are the intrinsic and extrinsic advantages computed via equation (3.4) for their respective reward streams. This implementation uses an episodic decay factor, d, that reduces intrinsic rewards as an episode progresses. RND prediction error bonuses are normalized by factoring the running standard deviation of discounted intrinsic returns, as done in [16]. The intrinsic reward at step t is computed as:

$$r_i = d^t \frac{\sum(\Phi(s_t) - \hat{\Phi}(s_t))^2}{n} \tag{4.10}$$

$$\tilde{r_i} = \frac{r_i}{\sigma\left[\sum_{k=0}^{t} \gamma_i^{t-k} r_i\right]} \tag{4.11}$$

where r_i is the un-normalized intrinsic reward, $\tilde{r_i}$ is the normalized intrinsic reward, n is the

output dimension of the RND predictor and target networks, σ is the standard deviation of the discounted return of target-prediction mean squared errors, and $d \in [0, 1]$ is the decay term.

The hyperparameters used for PPO with RND are shown in table 4.4. The RNDTillSeen agent also uses the same hyperparameters but does not receive intrinsic rewards once the target object is observed. The hyperparameters are chosen via trial and error; β_I and d required tuning to avoid pure exploration policies while still benefitting from the exploration bonus. While γ_I was kept to match the extrinsic discount factor, β_I was initially set to 0.2, and d was set to 1, which resulted in an agent that only explored. Reducing β_I to 0.1 somewhat reduced this behavior in early training interactions but again resulted in a pure exploration policy. Finally, d was set to 0.98, decaying intrinsic rewards as an episode progressed, and resulting in an agent that better balanced intrinsic and extrinsic rewards. Balancing the two reward streams is not trivial and is an interesting area of research. However; it is not further explored in this thesis.

Table 4.4: PPO and RND hyperparameters.

Hyperparameter	Value
Learning rate	3e-4
Learning rate decay	[3e-4, 3e-6]
Discount factor γ_E	0.99
Discount factor γ_I	0.99
Intrinsic decay rate d	0.98
β_E	1
β_I	0.1
GAE λ	0.95
# SGD epochs	4
Value function coefficient	0.5
Entropy coefficient	0.01
PPO clipping	0.1
Gradient clipping	0.5
Value function clipping	0.1
Rollout size	258
Batch size	8192
Minibatch size	512
Advantage normalization	Off
Max interactions	50M
# parallel workers	32

Figure 4.4 shows the average r_i over all environment interactions in a training batch of experience (equation (4.10)) for both RND and RNDTillSeen agents. This value decreases during training, showing that the RND predictor network is learning to predict the target network, but does not decay to zero. This behavior is desired as the agent must adapt its curiosity bonuses to new experiences while avoiding very low prediction errors that lead to an uncurious agent.

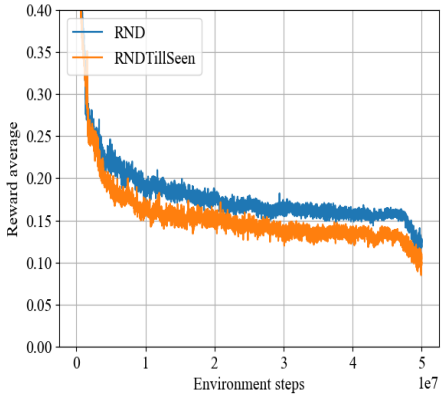

Figure 4.4: RND and RNDTillSeen average intrinsic rewards per training batch of experience.

Figure 4.5 shows the overall training dynamics and validation curves of the RGB ResNet18, RND, and RNDTillSeen agents. Both prediction error variants obtain greater success rates but lower SPL, indicating poorer trajectory efficiency. Both RND variants also demonstrate increasing trend lines, which suggests that more training interactions could lead to improved results. The RNDTillSeen agent achieves the largest validation success rates and SPL, but does not reach the same performance as either count-based or delta ExploreTillSeen agents presented in section 4.1. It suggests that rewarding ObjectNav agents for covering a greater area instead of observing dissimilar visual states provides greater benefits in ObjectNav.

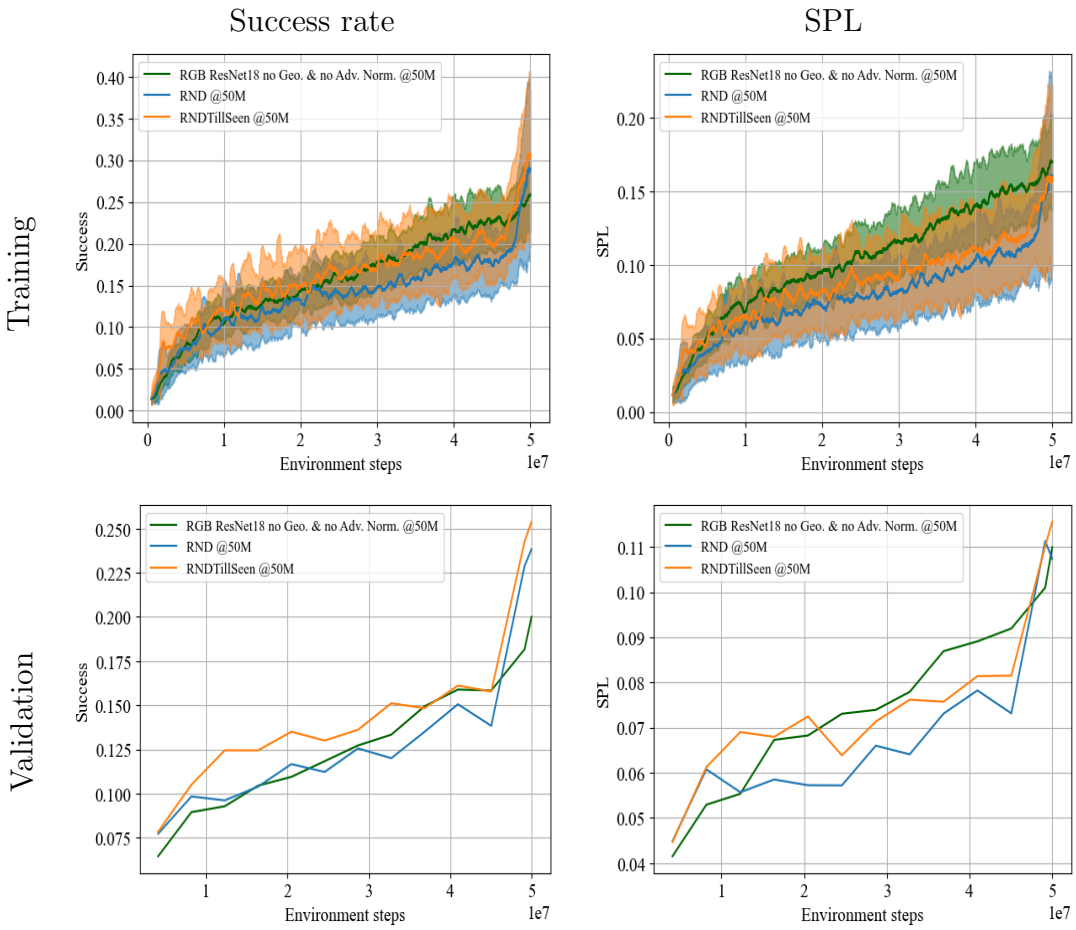

Figure 4.5: Training and validation curves of RGB ResNet18, RND, and RNDTillSeen agents.

Table 4.5 shows the performances of the prediction error agents over different object classes and difficulties. As with the delta and count ExploreTillSeen agents, the RND and RNDTillSeen agents demonstrate significant generalization errors overall (22.2% and 14.6% respectively).

Table 4.5: Performance metrics of the RND and RNDTillSeen agents trained for 50M environment interactions.

Object	RND		RNDTillSeen	
	Success rate	SPL	Success rate	SPL
	Train | Val	Train | Val	Train | Val	Train | Val
AlarmClock	0.2500 | 0.0733	0.2168 | 0.0338	0.0909 | 0.0333	0.0459 | 0.0186
Apple	0.5556 | 0.2667	0.3662 | 0.1600	0.3333 | 0.3667	0.2742 | 0.1653
BaseballBat	0.1667 | 0.0533	0.1614 | 0.0222	0.2000 | 0.0267	0.2000 | 0.0099
BasketBall	0.8000 | 0.4467	0.3670 | 0.1796	0.5000 | 0.5467	0.2600 | 0.2415
Bowl	0.3333 | 0.1200	0.2438 | 0.0669	0.4444 | 0.1000	0.2019 | 0.0627
GarbageCan	0.8333 | 0.5333	0.5864 | 0.2405	0.7500 | 0.6200	0.3190 | 0.2799
HousePlant	0.7273 | 0.4133	0.4483 | 0.1746	0.6667 | 0.4600	0.3519 | 0.2052
Laptop	0.4000 | 0.1733	0.2238 | 0.0847	0.4444 | 0.1800	0.1388 | 0.0882
Mug	0.3000 | 0.1467	0.1927 | 0.0676	0.1250 | 0.1067	0.0951 | 0.0701
SprayBottle	0.0000 | 0.0467	0.0000 | 0.0190	0.1250 | 0.0467	0.0588 | 0.0262
Television	0.6250 | 0.5133	0.3827 | 0.2063	0.7000 | 0.4600	0.4380 | 0.1810
Vase	0.1429 | 0.0733	0.1429 | 0.0328	0.2500 | 0.1000	0.2114 | 0.0385
Difficulty	Train | Val	Train | Val	Train | Val	Train | Val
Easy	0.6800 | 0.5632	0.3868 | 0.2247	0.6667 | 0.5522	0.2257 | 0.2114
Medium	0.5128 | 0.2209	0.3411 | 0.1135	0.4130 | 0.2353	0.2788 | 0.1233
Hard	0.2500 | 0.0663	0.1749 | 0.0280	0.1667 | 0.0995	0.1191 | 0.0470
All episodes	0.4600 | 0.2383	0.2927 | 0.1073	0.4000 | 0.2539	0.2181 | 0.1156

Table 4.6 compares the validation metrics of the baseline, RGB ResNet18, RND, and RNDTillSeen agents. As in table 4.3, the "Baseline @170" metrics in table 4.6 come from the RoboTHOR 2021 leaderboard. It shows the RND and RNDTillSeen agents improving validation success rates over the RGB ResNet18 (+3.8% and +5.4% respectively), with most improvements in easy and medium difficulty episodes. Although success rates improve for both of the long-term curiosity agents, the SPL barely improves, showing that these agents are less trajectory efficient. Still, both RND and RNDTillSeen agents improve validation success rates on both the baseline and RGB ResNet18 agents trained for 50M environment interactions.

Table 4.6: Validation performance comparison between baseline, RGB ResNet18 proposed in section 3.3.6, RND, RNDTillSeen agents.

Agent	Val success rate				Val SPL			
	Easy	Medium	Hard	Overall	Easy	Medium	Hard	Overall
Baseline @170M	—	—	—	0.3511	—	—	—	0.1737
Baseline @50M	0.2390	0.1032	0.0746	0.1211	0.1212	0.0574	0.0449	0.0661
RGB ResNet18 no Geo. & no Adv.Norm @50M	0.4698	0.1801	0.0647	0.2000	0.2629	0.1007	0.0303	0.1099
RND @50M	0.5632	0.2209	0.0663	0.2383	0.2247	0.1135	0.0280	0.1073
RNDTillSeen @50M	0.5522	0.2353	0.0995	0.2539	0.2114	0.1233	0.0470	0.1156

Both prediction error bonus schemes respectively reach 68% and 72% of the leaderboard baseline's validation success rate but with less than a third of the training experience. Given the increasing trend in training and validation curves shown in figure 4.5, it would still need to be evaluated whether this gap in performance could be bridged with more training experience.

Figure 4.6 demonstrates two sample trajectories of the RNDTillSeen agent in validation scenarios. It shows the agent is capable of locating the Apple but not the Television, despite the Television validation success rate shown in table 4.5 being greater than that of the Apple. The trajectories show that the agent is capable of exploratory behavior but can still be either ineffective at exploring the available area of the scene, or experience difficulty distinguishing target objects from other object classes.

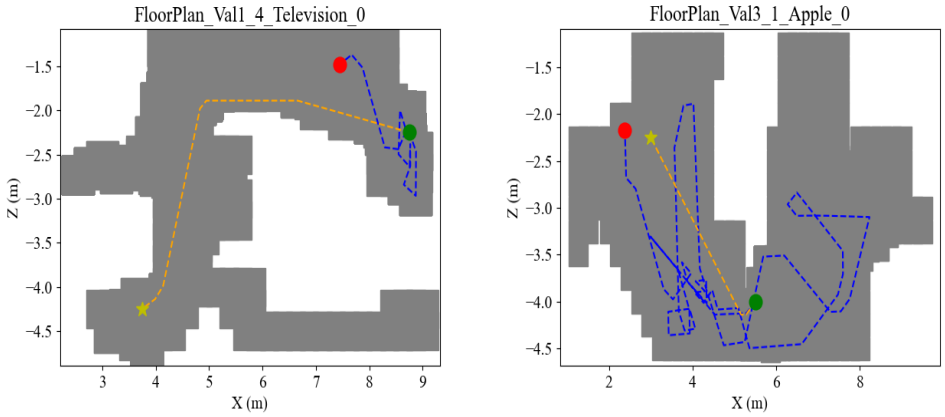

Figure 4.6: Television (left) and Apple (right) sample trajectories of the RNDTillSeen agent. Green and red dots indicate starting and end points respectively, stars indicate target location, blue lines indicate agent trajectories, and orange lines indicate the shortest path.

These experiments demonstrate that RND and RNDTillSeen can improve performance in sparse reward ObjectNav settings, but not to the same extent as the delta and count ExploreTillSeen reward schemes presented in section 4.1. A possible explanation for this may be due to the magnitude of intrinsic rewards and the actions that are reinforced. The delta ExploreTillSeen agent only receives exploration bonuses when moving forward, while the count-based ExploreTillSeen agent receives a bonus for moving ahead, and a smaller bonus for rotation actions due to the occupied cell's count being incremented. The RND agent receives bonuses for both types of actions, with magnitudes that vary depending on the RND prediction error. It remains a field of research to investigate how to optimally balance and scale intrinsic and extrinsic rewards in RL.

4.3 Episodic and Long-term Novelty

Although the episodic bonuses presented in section 4.1 allow better performances than the RND intrinsic rewards, both methods still show success rate improvements over the RGB ResNet18 proposed in section 3.3.6. While RND bonuses change as the agent observes novel states, both count-based and delta ExploreTillSeen reward schemes reinforce the agent independently of the quantity or distribution of environment observations. Combining these two streams of exploration rewards in ObjectNav could couple the benefit of an area coverage bonus with any improvements obtained by long-term novelty bonuses.

As mentioned in section 2.2.4, the Agent57 [19] and never give up (NGU) [18] agents demonstrate that incorporating episodic and long-term novelty enhance the agent's performances in very sparse reward settings. These approaches use other techniques, like the off-policy recurrent replay distributed deep Q-network (R2D2) [88], with multiple policies that differ in the magnitude of intrinsic rewards received. Still, their formulation of intrinsic rewards enables exploratory agents to succeed in difficult sparse reward games. These agents modulate episodic rewards, which are formulated as a count approximation using K-nearest neighbors (KNN), with RND prediction errors. Equations (2.53), (2.54) and (2.56) define the overall intrinsic rewards.

The agent proposed in this section uses a similar approach to combining episodic and long-term novelty rewards, but modifies how episodic rewards are calculated: KNN episodic rewards are replaced with count-based ones. This modification means episodic rewards are computationally efficient and don't require keeping an episodic memory buffer. Furthermore, the combined exploration bonuses are adapted to propose a new "NGUTillSeen"

reward scheme for ObjectNav which extends equation (4.7) as

$$r = r_{slack} + \begin{cases} r_i & \text{until goal is seen} \\ r_{goal\_seen} & \text{first time goal is seen} \\ -\Delta_{geo} + S r_{success} & \text{otherwise} \end{cases} \tag{4.12}$$

$$r_i = r_{count} \cdot \min(\max(\alpha, \alpha_{min}), \alpha_{max}) \tag{4.13}$$

$$\alpha = 1 + \frac{err(\Phi(s_t), \hat{\Phi}(s_t)) - \mu_{err}}{\sigma_{err}} \tag{4.14}$$

$$err(\Phi(s_t), \hat{\Phi}(s_t)) = \frac{\sum(\Phi(s_t) - \hat{\Phi}(s_t))^2}{n} \tag{4.15}$$

where r_{count} is defined by equation (4.6), err is the mean squared RND prediction error, μ_{err} and σ_{err} are the mean and standard deviation of the prediction error, Φ and $\hat{\Phi}$ are the RND target and predictor networks, n is the output dimension of the RND predictor and target networks, α modulates the episodic r_{count} reward, and α_{max} and α_{min} are the maximum and minimum α values.

The NGUTillSeen agent is trained following the same network architecture used for the RND and RNDTillSeen agents, and the hyperparameter configuration is detailed in table 4.7. Notable differences are the absence of the RND decay rate described in section 4.2, and the intrinsic and extrinsic rewards are equally weighted (β_I, β_E). Similar to the original implementations of [18, 19], α is normalized by its running mean and standard deviation over training. This is different to the RND bonus normalization method, which standardizes intrinsic rewards by factoring the standard deviation of discounted intrinsic returns.

Table 4.7: PPO and NGUTillSeen hyperparameters

Hyperparameter	Value
Learning rate	3e-4
Learning rate decay	[3e-4, 3e-6]
Discount factor γ_E	0.99
Discount factor γ_I	0.99
β_E	1
β_I	1
α_{max}	5
α_{min}	1
r_{expl}	5
GAE λ	0.95
# SGD epochs	4
Value function coefficient	0.5
Entropy coefficient	0.01
PPO clipping	0.1
Gradient clipping	0.5
Value function clipping	0.1
Rollout size	258
Batch size	8192
Minibatch size	512
Advantage normalization	Off
Max interactions	50M
# parallel workers	32

Figure 4.7 and figure 4.8 illustrate the count-based ExploreTillSeen and NGUTillSeen agents' performances throughout training and validation. While figure 4.7 charts overall metrics, figure 4.8 charts those metrics for each episode difficulty category. Although the NGUTillSeen agent learns easy, medium and hard scenarios faster in the short run, the

count-based ExploreTillSeen agent achieves better final validation metrics. Additionally, the NGUTillSeen agent obtains greater validation SPL throughout training, but eventually reaches a similar rate as the count-based ExploreTillSeen agent.

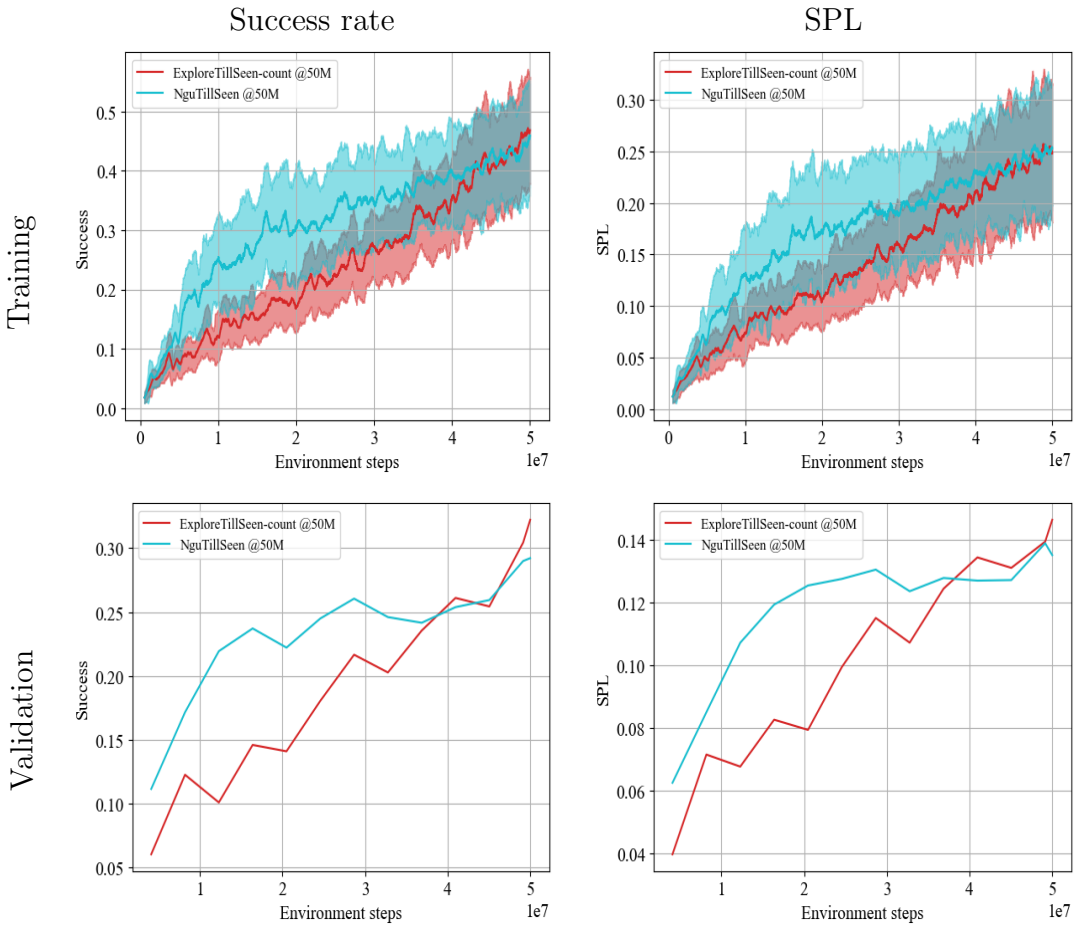

Figure 4.7: Training and validation curves of count-based ExploreTillSeen and NGUTillSeen agents.

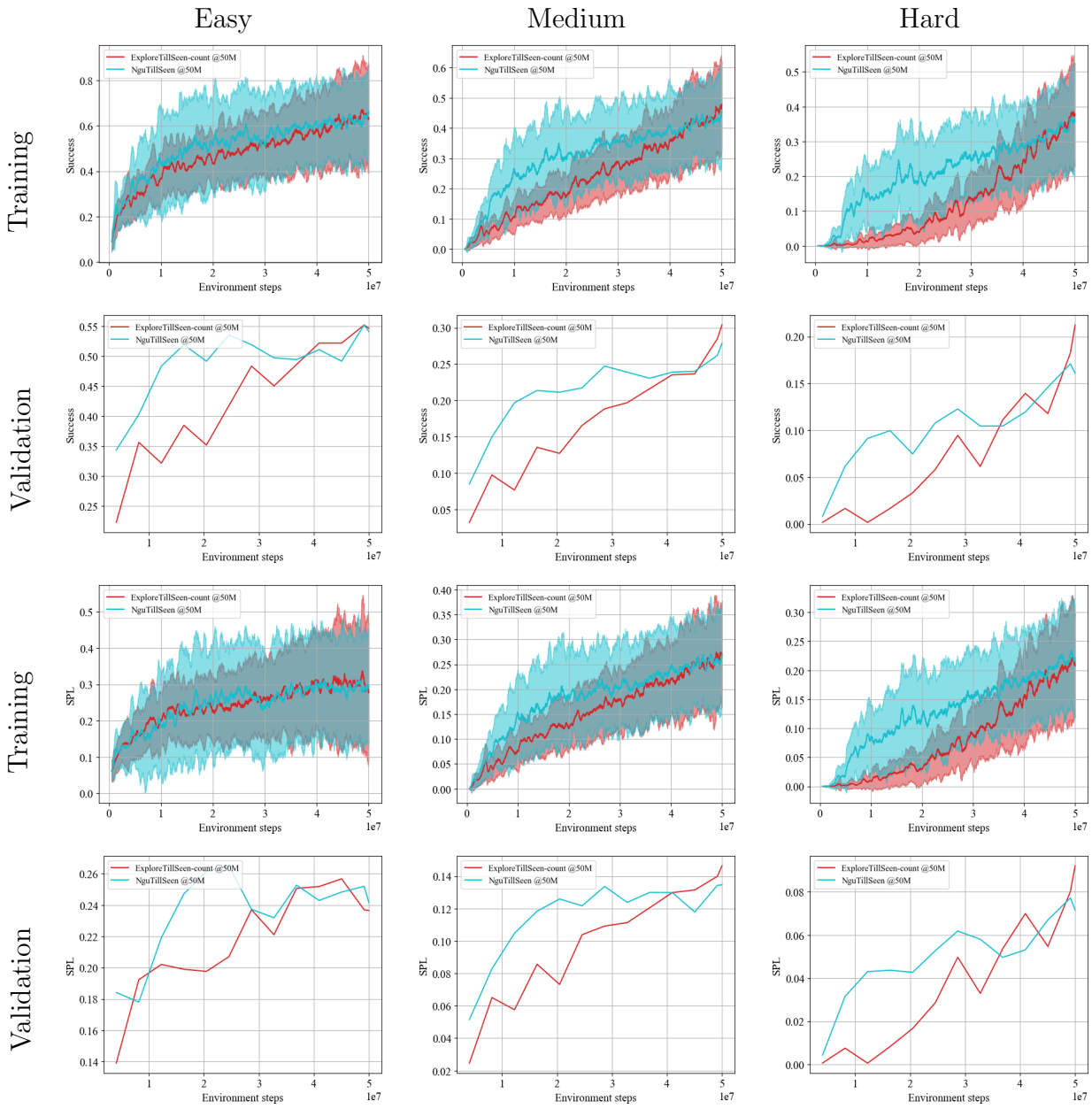

Figure 4.8: Count-based ExploreTillSeen and NGUTillSeen agents' training and validation curves for each episode difficulty.

Table 4.8 shows the NGUTillSeen agent's performance metrics across object categories and episode difficulties. As with the ExploreTillSeen-count agent, the NGUTillSeen agent shows a large gap between validation and training success rates (30.2%). It also shows success rates of 100% on the training batch for GarbageCan and Television object classes, but only 60% and 62.7% validation success for each object respectively. Hence, it is clear that generalization errors remains an issue for this agent.

Table 4.8: Performance metrics of the NGUTillSeen agent trained for 50M environment interactions.

Object	Train success rate	Train SPL	Val success rate	Val SPL
AlarmClock	0.1667	0.0558	0.0667	0.0485
Apple	0.7778	0.3510	0.4600	0.2114
BaseballBat	0.0000	0.0000	0.0400	0.0361
BasketBall	0.9167	0.4966	0.5133	0.1701
Bowl	0.4444	0.2868	0.0933	0.0424
GarbageCan	1.0000	0.5299	0.6000	0.2829
HousePlant	0.8889	0.5218	0.6133	0.2827
Laptop	0.6667	0.3850	0.2533	0.1209
Mug	0.1667	0.0000	0.1600	0.1289
SprayBottle	0.1429	0.0000	0.0467	0.0361
Television	1.0000	0.5973	0.6267	0.2300
Vase	0.1667	0.1667	0.0333	0.0317
Difficulty	**Train success rate**	**Train SPL**	**Val success rate**	**Val SPL**
Easy	0.7368	0.2624	0.5412	0.2415
Medium	0.6842	0.4047	0.2785	0.1347
Hard	0.4545	0.2707	0.1609	0.0716
Total	0.5941	0.3196	0.2922	0.1351

Table 4.9 compares validation performance metrics between the baseline, RGB ResNet18 (from section 3.3.6), count-based ExploreTillSeen, and NGUTillSeen agents. Note that the "Baseline @170" metrics in table 4.9 come from the RoboTHOR 2021 leaderboard, while "Baseline @50" refers to the agent presented in section 3.3.7. Both exploration agents perform similarly in episodes within the 20th percentile of shortest path lengths, but the count-based ExploreTillSeen agent performs better in sparser reward episodes.

The NGUTillSeen agent comes close to the leaderboard baseline benchmark for validation success rate (less than 6%) and shows an increasing trend in training and validation performance curves. Hence, it is possible that extending this agent's training further could reach or surpass the leaderboard baseline's validation success rate.

Table 4.9: Validation performance comparison between baseline, RGB ResNet18 proposed in section 3.3.6, ExploreTillSeen-count, and NGUTillSeen agents.

Agent	Val success rate				Val SPL			
	Easy	Medium	Hard	Overall	Easy	Medium	Hard	Overall
Baseline @170M	—	—	—	0.3511	—	—	—	0.1737
Baseline @50M	0.2390	0.1032	0.0746	0.1211	0.1212	0.0574	0.0449	0.0661
RGB ResNet18 no Geo. & no Adv.Norm @50M	0.4698	0.1801	0.0647	0.2000	0.2629	0.1007	0.0303	0.1099
ExploreTillSeen-count @50M	0.5467	0.3037	0.2123	0.3222	0.2365	0.1464	0.0921	0.1464
NGUTillSeen @50M	0.5412	0.2785	0.1609	0.2922	0.2415	0.1347	0.0716	0.1351

Figure 4.9 demonstrates two sample trajectories of the NGUTillSeen agent in validation scenarios. It shows that this agent is capable of locating the Television but not the Apple. Furthermore, the agent demonstrates more direct trajectories compared to the ExploreTillSeen-count agent's trajectories shown in figure 4.2.

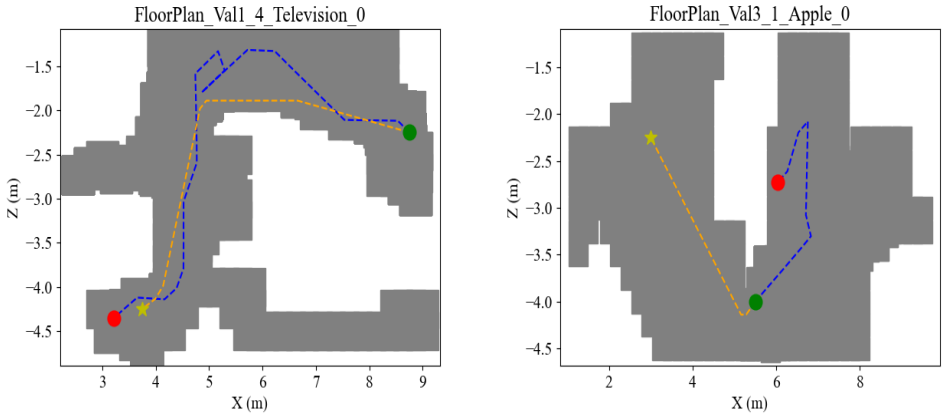

Figure 4.9: Television (left) and Apple (right) sample trajectories of the NGUTillSeen agent. Green and red dots indicate starting and end points respectively, stars indicate target location, blue lines indicate agent trajectories, and orange lines indicate the shortest path.

4.4 Summary

This section presented agents trained on reward schemes that incorporate exploration bonuses to help attain the sparse rewards present in the ObjectNav setting. Episodic bonuses were employed via the delta and count-based ExploreTillSeen reward schemes, which substantially improve upon the RGB ResNet18 agent proposed in section 3.3.6. The count-based variant ultimately achieves a validation success rate within 3% of the original AllenAct baseline while training for less than a third of the environment interactions. In addition to episodic intrinsic rewards, RND prediction error bonuses were also explored. Although they do improve validation success over the RGB ResNet18 agent, they do not provide as much benefit when compared to the schemes employing episodic intrinsic rewards. Tuning RND and RNDTillSeen agents to appropriately balance intrinsic

and extrinsic rewards is not trivial. If the intrinsic reward discount factor, γ_I, weight, β_I, or decay, d, are too large, the agent's policy becomes purely exploratory and fails to obtain extrinsic rewards. Hence, research towards appropriately balancing these two reward streams would enable greater use of the RND prediction error bonuses for this and other RL tasks. Finally, the NGUTillSeen reward scheme, which employs both episodic visitation counts and long-term novelty via an RND prediction error, obtains a 3% lower success rate when compared to the count-based ExploreTillSeen agent. This is despite the agent presenting more exploratory behaviors initially during training. Nonetheless, all methods demonstrate improved success over the RGB ResNet18 proposed in section 3.3.6, and the baseline trained for the same amount of environment interactions (50M). This shows that they are valid for dealing with sparse rewards in the ObjectNav challenge to varying degrees. The results in this chapter suggest that, of the exploration bonus methods presented that deal with sparse rewards, the count-based ExploreTillSeen scheme is best suited for the ObjectNav task.

Chapter 5: Conclusion

5.1 Summary

This thesis presents on-policy agents trained on the ObjectNav task and provides a summary of the barriers encountered. An ablation study on the baseline approach to the RoboTHOR ObjectNav challenge is presented. Insights gained from the study guide the design of an ObjectNav agent that improves generalization by removing dense geodesic rewards, and reduces the processing time by limiting sensing modalities to RGB. The empirical justification and evaluation of the new agent configuration is documented and shown to improve validation performance when compared to the baseline. Then, to ameliorate the challenge of sparse rewards inherent in the task, this agent is coupled with reward schemes that use episodic and long-term novelty bonuses to train exploratory agents. These proposed reward schemes build upon an ExploreTillSeen [21] reward schedule, but simplify some of the computations to reduce processing cost per environment interaction. These simplifications are original to this work and enable agents without semantic segmentation to use the proposed reward schemes.

In summary, the proposed count-based ExploreTillSeen reward scheme outperformed all other agents presented in this thesis and is roughly 3% shy of the RoboTHOR challenge's baseline success rate[1]. Although this performance is worse than the original baseline, it is achieved while only having to process a third of the overall training steps. Furthermore, the increasing trend in training and validation curves suggests that further training could bridge

[1]According to the RGB+D ResNet18-ImageNet submission's validation success rate from https://leaderboard.allenai.org/robothor_objectnav/submissions/public

or surpass this gap. The episodic exploration bonuses result in more performant agents than long-term prediction error bonuses via random network distillation (RND); however, both types of intrinsic rewards demonstrate improvements over the baseline and the RGB ResNet18 agents trained for 50 million environment interactions. The NGUTillSeen agent proposed in this work achieves greater performance in validation scenes in the short run, but finally comes within 3% success rate of the count-based ExploreTillSeen approach, and 6% of the baseline's leaderboard entry for the RoboTHOR ObjectNav challenge.

5.2 Contributions

The main contributions of this work are:

- The development and evaluation of a lighter (i.e., less hardware resource intensive) RGB ResNet18 agent for on-policy ObjectNav in the RoboTHOR environment that improves success rates in validation scenes (i.e. unseen environments).
- The design and empirical justification of uniquely adapted ExploreTillSeen reward schemes for ObjectNav that improve the exploration and performance of the modified baseline. Those schemes are:
  - delta and count-based ExploreTillSeen
  - RNDTillSeen
  - NGUTillSeen

- The development and application of a RND agent for ObjectNav that augments the RGB ResNet18 agent to improve the overall validation success rate. The proposed RND agent is the first demonstration of prediction error bonuses applied to object goal navigation that shows a benefit in validation success.

5.3 Limitations

One limitation is the number of environment interactions allowed per experiment conducted. ObjectNav approaches commonly use more environment interactions when training agents. For example, the leaderboard entry for the RoboTHOR challenge baseline agent reaches peak success rate after 170M environment interactions. In contrast, the most environment interactions used to train an agent presented in this thesis is 50M environment interactions. Since many agents demonstrated increasing trends in their learning dynamics, the peak possible performance metrics of a given agent configuration may not have been reached. This would be worth validating in future work on computing systems with greater hardware resources.

The number of runs per experiment is another limitation in the experiments conducted. As mentioned in section 2.3.2, machine learning configurations are normally run for multiple iterations and their averages are reported to account for variance between runs; however, due to constraints on time and the processing requirements of the simulated environment, single runs are reported. Leaderboard benchmarks and ObjectNav approaches often report on their best performing agents. As such, the experiments conducted in this thesis are still valid to compare against challenge leaderboard benchmarks.

5.4 Future Work

Addressing Generalization and Feature Representation

While this work demonstrates improvements in obtaining sparse rewards in the ObjectNav task, it is but one of the problems to be addressed. Representing observations in a useful manner plays an important role. As demonstrated in [34], simply changing the RGB feature extraction layer improves test success rates from 26% to 47%. Deitke et al. [77] show further improvements to generalization by increasing the dataset size. However, these approaches rely on the reduced change in distance between the agent and its target object to deal with sparse rewards. As such, the exploration bonus methods presented in this thesis should be coupled with approaches to improve generalization and state representation to aggregate their benefits.

Balancing Intrinsic and Extrinsic Rewards

As demonstrated in this thesis, there is a benefit in combining intrinsic and extrinsic rewards for the ObjectNav task; although, grid-searching hyperparameters is neither optimal nor time-efficient to achieve this. The different reward stream weights and discount factors affect the tradeoff between exploration and exploitation. Methods such as population-based training (PBT) [89] and population-based bandits (PB2) [90] efficiently search for hyperparameters in a number of machine learning tasks and are likely beneficial to balancing intrinsic and extrinsic reward related hyperparameters. Agent57 and NGU are two approaches that train a set of policies that vary in discount factors and intrinsic reward

weights. This enables their approaches to discover policies that adequately explore in a given task. This remains a field to explore in the context of object goal navigation.

Transfer to Real Environments

This thesis looked to address reward sparsity in the RoboTHOR simulated environment but did not address the problem of transferring one of the trained agents to the physical world. The RoboTHOR and Habitat challenges attempt to minimize the potential sim-2-real gap by building simulated environments modeled from real world equivalents. However, it is not expected that this problem is fully mitigated in these challenges; hence, the sim-2-real gap should still be addressed. One way to achieve this is by using a domain randomization technique, such as automatic domain randomization [3]. This method samples different simulator physics parameters and perturbs the agent's actions and observations such that the learnt policy is robust to the unmodeled characteristics of the real world environment.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems (NIPS-20)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[2] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames," in *Eighth International Conference on Learning Representations (ICLR-20)*, Jan. 2020.

[3] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's Cube with a Robot Hand," *arXiv:1910.07113 [cs, stat]*, Oct. 2019, arXiv: 1910.07113. [Online]. Available: http://arxiv.org/abs/1910.07113

[4] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *International Conference on Computer Vision (ICCV-19)*. Seoul, Korea (South): IEEE, Oct. 2019, pp. 9338–9346. [Online]. Available: https://ieeexplore.ieee.org/document/9010745/

[5] M. Deitke, W. Han, A. Herrasti, A. Kembhavi, E. Kolve, R. Mottaghi, J. Salvador, D. Schwenk, E. VanderBilt, M. Wallingford, L. Weihs, M. Yatskar, and A. Farhadi, "RoboTHOR: An Open Simulation-to-Real Embodied AI Platform," in *Conference on Computer Vision and Pattern Recognition (CVPR-20)*. IEEE/CVF, June 2020.

[6] D. Ha and J. Schmidhuber, "Recurrent World Models Facilitate Policy Evolution," in *Advances in Neural Information Processing Systems (NIPS-18)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/2de5d16682c3c35007e4e92982f1a2ba-Paper.pdf

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *33rd International Conference on Machine Learning (ICML-16)*, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1928–1937. [Online]. Available: https://proceedings.mlr.press/v48/mniha16.html

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347 [cs]*, Aug. 2017, arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347

[9] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[10] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss,

I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019. [Online]. Available: http://www.nature.com/articles/s41586-019-1724-z

[11] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aar6404

[12] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic Motivation Systems for Autonomous Mental Development," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 2, pp. 265–286, Apr. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4141061/

[13] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning," in *Conference on Computational Intelligence and Games*. Santorini, Greece: IEEE, Sep 2016, pp. 341–348, the Best Paper Award.

[14] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying Count-Based Exploration and Intrinsic Motivation," in *Advances*

in Neural Information Processing Systems (NIPS-16), D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/afda332245e2af431fb7b672a68b659d-Paper.pdf

[15] J. Choi, Y. Guo, M. Moczulski, J. Oh, N. Wu, M. Norouzi, and H. Lee, "Contingency-Aware Exploration in Reinforcement Learning," in *International Conference on Learning Representations (ICLR-19)*, 2019. [Online]. Available: https://openreview.net/forum?id=HyxGB2AcY7

[16] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," in *International Conference on Learning Representations (ICLR-19)*, 2019. [Online]. Available: https://openreview.net/forum?id=H1lJJnR5Ym

[17] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-Driven Exploration by Self-Supervised Prediction," in *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW-17)*. Honolulu, HI, USA: IEEE, Jul. 2017, pp. 488–489. [Online]. Available: http://ieeexplore.ieee.org/document/8014804/

[18] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, and C. Blundell, "Never Give Up: Learning Directed Exploration Strategies," in *International Conference on Learning Representations (ICLR-20)*, 2020. [Online]. Available: https://openreview.net/forum?id=Sye57xStvB

[19] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari Human Benchmark," in

37th International Conference on Machine Learning (ICML-20), H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 507–517. [Online]. Available: https://proceedings.mlr.press/v119/badia20a.html

[20] J. Ye, D. Batra, A. Das, and E. Wijmans, "Auxiliary Tasks and Exploration Enable ObjectGoal Navigation," in *International Conference on Computer Vision (ICCV-21)*. IEEE/CVF, 2021, pp. 16 097–16 106.

[21] O. Maksymets, V. Cartillier, A. Gokaslan, E. Wijmans, W. Galuba, S. Lee, and D. Batra, "THDA: Treasure Hunt Data Augmentation for Semantic Navigation," in *International Conference on Computer Vision (ICCV-21)*. Montreal, QC, Canada: IEEE, Oct. 2021, pp. 15 354–15 363. [Online]. Available: https://ieeexplore.ieee.org/document/9711292/

[22] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[23] Y. LeCun and Y. Bengio, *Convolutional Networks for Images, Speech, and Time Series*. Cambridge, MA, USA: MIT Press, 1998, p. 255–258.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems (NIPS-12)*, vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[25] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015, pp. 1–14.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Conference on Computer Vision and Pattern Recognition (CVPR-16)*. Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 770–778. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Conference on Computer Vision and Pattern Recognition (CVPR-14)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2014, pp. 580–587. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2014.81

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems (NIPS-15)*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[29] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *International Conference on Computer Vision (ICCV-17)*. IEEE, 2017, pp. 2980–2988.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet

Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015. [Online]. Available: http://link.springer.com/10.1007/s11263-015-0816-y

[31] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," in *38th International Conference on Machine Learning (ICML-21)*, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html

[32] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[33] R. Ramrakhya, E. Undersander, D. Batra, and A. Das, "Habitat-Web: Learning Embodied Object-Search Strategies from Human Demonstrations at Scale," in *Conference on Computer Vision and Pattern Recognition (CVPR-22)*. New Orleans, LA, USA: IEEE, Jun. 2022, pp. 5163–5173. [Online]. Available: https://ieeexplore.ieee.org/document/9880429/

[34] A. Khandelwal, L. Weihs, R. Mottaghi, and A. Kembhavi, "Simple but Effective: CLIP Embeddings for Embodied AI," in *Conference on Computer Vision and Pattern Recognition (CVPR-21)*. IEEE/CVF, 2021, pp. 14 809–14 818.

[35] J. L. Elman, "Finding Structure in Time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1207/s15516709cog1402_1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1

[36] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997.

[37] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, Sep. 2014.

[38] D. S. Chaplot, D. P. Gandhi, A. Gupta, and R. R. Salakhutdinov, "Object goal navigation using goal-oriented semantic exploration," in *Advances in Neural Information Processing Systems (NIPS-20)*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 4247–4258. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/2c75cf2681788adaca63aa95ae028b22-Paper.pdf

[39] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," Sep. 2017, arXiv:1703.03864 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1703.03864

[40] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, Mar. 1951. [Online]. Available: http://projecteuclid.org/euclid.aoms/1177729694

[41] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[42] N. Qian, "On the Momentum Term in Gradient Descent Learning Algorithms," *Neural Networks*, vol. 12, pp. 145–151, Jan. 1999. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0893608098001166

[43] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-24, Mar 2010. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html

[44] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference on Learning Representations, (ICLR-15)*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[45] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, 2nd ed. Cambridge, Massachusetts: The MIT Press, 2018.

[46] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," in *32nd International Conference on Machine Learning (ICML-15)*, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 1889–1897. [Online]. Available: https://proceedings.mlr.press/v37/schulman15.html

[47] G. A. Rummery and M. Niranjan, "On-Line Q-Learning Using Connectionist Systems," Cambridge University Engineering Department, Cambridge, England, Tech. Rep. TR 166, 1994.

[48] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *Deep Learning Workshop at Neural Information Processing Systems (NIPS-13)*, Dec. 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[50] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*. AAAI Press, 2016, p. 2094–2100.

[51] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," in *33rd International Conference on Machine Learning (ICML-16)*, vol. 48. JMLR.org, 2016, pp. 1995–2003.

[52] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," in *4th International Conference on Learning Representations, (ICLR-16)*, Y. Bengio and Y. LeCun, Eds., May 2016. [Online]. Available: http://arxiv.org/abs/1506.02438

[53] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, May 1992. [Online]. Available: http://link.springer.com/10.1007/BF00992696

[54] R. S. Sutton, "Integrated Modeling and Control Based on Reinforcement Learning and Dynamic Programming," in *Advances in Neural Information Processing Systems (NIPS-90)*, vol. 3. Morgan-Kaufmann, 1990. [Online]. Available: https://proceedings.neurips.cc/paper/1990/hash/d9fc5b73a8d78fad3d6dffe419384e70-Abstract.html

[55] M. G. Bellemare, J. Veness, G. Deepmind, and E. Talvitie, "Skip context tree switching," in *31st International Conference on Machine Learning (ICML-14)*, 2014.

[56] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-Based Exploration with Neural Density Models," in *34th International Conference on Machine Learning (ICML-17) - Volume 70*. JMLR.org, 2017, p. 2721–2730.

[57] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional Image Generation with PixelCNN Decoders," in *30th International Conference on Neural Information Processing Systems (NIPS-16)*. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 4797–4805.

[58] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems (NIPS-17)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates,

Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3a20f62a0af1aa152670bab3c602feed-Paper.pdf

[59] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. Efros, "Large-scale study of curiosity-driven learning," in *7th International Conference on Learning Representations (ICLR 2019)*, May 2019, pp. 1–17, seventh International Conference on Learning Representations, ICLR 2019 ; Conference date: 06-05-2019 Through 09-05-2019. [Online]. Available: https://iclr.cc/

[60] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3D: Learning from RGB-D Data in Indoor Environments," *International Conference on 3D Vision (3DV)*, 2017.

[61] S. K. Ramakrishnan, A. Gokaslan, E. Wijmans, O. Maksymets, A. Clegg, J. M. Turner, E. Undersander, W. Galuba, A. Westbury, A. X. Chang, M. Savva, Y. Zhao, and D. Batra, "Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. [Online]. Available: https://openreview.net/forum?id=-v4OuqNs5P

[62] Unity Technologies, "Unity." [Online]. Available: https://unity.com/

[63] D. Batra, A. Gokaslan, A. Kembhavi, O. Maksymets, R. Mottaghi, M. Savva, A. Toshev, and E. Wijmans, "ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects," *arXiv:2006.13171 [cs]*, Aug. 2020, arXiv: 2006.13171. [Online]. Available: http://arxiv.org/abs/2006.13171

[64] J. A. Sethian, "Fast Marching Methods," *SIAM Review*, vol. 41, no. 2, pp. 199–235, Jan. 1999. [Online]. Available: http://epubs.siam.org/doi/10.1137/S0036144598347059

[65] J. Jiang, L. Zheng, F. Luo, and Z. Zhang, "RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation," Aug. 2018, arXiv:1806.01054 [cs]. [Online]. Available: http://arxiv.org/abs/1806.01054

[66] Z. D. Guo, M. G. Azar, B. Piot, B. A. Pires, and R. Munos, "Neural Predictive Belief Representations," *arXiv:1811.06407 [cs, stat]*, Aug. 2019, arXiv: 1811.06407. [Online]. Available: http://arxiv.org/abs/1811.06407

[67] Z. D. Guo, B. A. Pires, B. Piot, J.-B. Grill, F. Altché, R. Munos, and M. G. Azar, "Bootstrap Latent-Predictive Representations for Multitask Reinforcement Learning," in *37th International Conference on Machine Learning (ICML-20)*. PMLR, Nov. 2020, pp. 3875–3886, iSSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v119/guo20g.html

[68] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," in *35th International Conference on Machine Learning (ICML-18)*, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1407–1416. [Online]. Available: https://proceedings.mlr.press/v80/espeholt18a.html

[69] H. Luo, A. Yue, Z.-W. Hong, and P. Agrawal, "Stubborn: A Strong Baseline for Indoor Object Navigation," in *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ, 2022, pp. 3287–3293.

[70] E. Wijmans, S. Datta, O. Maksymets, A. Das, G. Gkioxari, S. Lee, I. Essa, D. Parikh, and D. Batra, "Embodied Question Answering in Photorealistic Environments With Point Cloud Perception," in *Conference on Computer Vision and Pattern Recognition (CVPR-19)*. Long Beach, CA, USA: IEEE, Jun. 2019, pp. 6652–6661. [Online]. Available: https://ieeexplore.ieee.org/document/8954388/

[71] S. K. Ramakrishnan, D. S. Chaplot, Z. Al-Halah, J. Malik, and K. Grauman, "PONI: Potential Functions for ObjectGoal Navigation with Interaction-free Learning," in *Conference on Computer Vision and Pattern Recognition (CVPR-22)*. IEEE/CVF, 2022, pp. 18 868–18 878.

[72] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[73] V. Cartillier, Z. Ren, N. Jain, S. Lee, I. Essa, and D. Batra, "Semantic MapNet: Building Allocentric Semantic Maps and Representations from Egocentric Views," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, May 2021, pp. 964–972. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16180

[74] A. Eftekhar, A. Sax, J. Malik, and A. Zamir, "Omnidata: A Scalable Pipeline for Making Multi-Task Mid-Level Vision Datasets from 3D Scans," in *International Conference on Computer Vision (ICCV-21)*. IEEE/CVF, 2021, pp. 10 766–10 776.

[75] M. Caron, H. Touvron, I. Misra, H. Jegou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging Properties in Self-Supervised Vision Transformers," in *International Conference on Computer Vision (ICCV-21)*. Montreal, QC, Canada: IEEE, Oct. 2021, pp. 9630–9640. [Online]. Available: https://ieeexplore.ieee.org/document/9709990/

[76] L. Weihs, J. Salvador, K. Kotar, U. Jain, K.-H. Zeng, R. Mottaghi, and A. Kembhavi, "AllenAct: A Framework for Embodied AI Research," *arXiv preprint arXiv:2008.12760*, 2020.

[77] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, K. Ehsani, J. Salvador, W. Han, E. Kolve, A. Kembhavi, and R. Mottaghi, "ProcTHOR: Large-Scale Embodied AI Using Procedural Generation," in *Advances in Neural Information Processing Systems (NIPS-22)*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=4-bV1bi74M

[78] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying Generalization in Reinforcement Learning," in *36th International Conference on Machine Learning (ICML-19)*, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1282–1289. [Online]. Available: https://proceedings.mlr.press/v97/cobbe19a.html

[79] K. Yadav, R. Ramrakhya, A. Majumdar, V.-P. Berges, S. Kuhar, D. Batra, A. Baevski, and O. Maksymets, "Offline Visual Representation Learning for Embodied Navigation," Apr. 2022, arXiv:2204.13226 [cs]. [Online]. Available: http://arxiv.org/abs/2204.13226

[80] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, "What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study," in *International Conference on Learning Representations (ICLR-21)*, 2021. [Online]. Available: https://openreview.net/forum?id=nIAxjsniDzg

[81] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *35th International Conference on Machine Learning (ICML-18)*, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3053–3062. [Online]. Available: https://proceedings.mlr.press/v80/liang18b.html

[82] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016. [Online]. Available: https://github.com/openai/gym

[83] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems (NIPS-19)*, H. Wallach, H. Larochelle,

A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[84] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A System for Large-Scale Machine Learning." in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI-16)*, vol. 16, 2016, pp. 265–283.

[85] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay," *arXiv:1803.09820 [cs, stat]*, Apr. 2018, arXiv: 1803.09820. [Online]. Available: http://arxiv.org/abs/1803.09820

[86] R. Gulde, M. Tuscher, A. Csiszar, O. Riedel, and A. Verl, "Deep Reinforcement Learning using Cyclical Learning Rates," in *Third International Conference on Artificial Intelligence for Industries (AI4I-20)*, 2020, pp. 32–35.

[87] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[88] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan, and R. Munos, "Recurrent Experience Replay in Distributed Reinforcement Learning," in *International Conference on Learning Representations (ICLR-19)*, 2019. [Online]. Available: https://openreview.net/forum?id=r1lyTjAqYX

[89] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population Based Training of Neural Networks," Nov. 2017, arXiv:1711.09846 [cs]. [Online]. Available: http://arxiv.org/abs/1711.09846

[90] J. Parker-Holder, V. Nguyen, and S. J. Roberts, "Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits," in *Advances in Neural Information Processing Systems (NIPS-20)*, vol. 33. Curran Associates, Inc., 2020, pp. 17 200–17 211. [Online]. Available: https://papers.nips.cc/paper/2020/hash/c7af0926b294e47e52e46cfebe173f20-Abstract.html

[91] Y. Mao, "Intro to Deep Learning: Module 5," University Lecture, University of Ottawa, 2019.

APPENDICES

Appendix A: Neural Network Architectures

A.1 Convolutional Neural Networks

A.1.1 LeNet

The convolutional neural network is described in Lecun et al. [23]. The convolution operation filters an input feature map with a C by k_1 by k_2 kernel where C is determined by the channels of the input feature map and the other dimensions are a design parameter. A kernel is learnt per output feature map, which is computed by sliding each over the volume of input features. Sub-sampling operations commonly used between convolutional layers are max, min, or mean pooling.

A.1.2 ResNet

The ResNet architecture introduces residual and bottleneck blocks. Residual blocks consist of two 3x3 convolutional layers, each outputting 64 feature maps, and residual connections. Similarly, bottleneck blocks compress input feature maps with a 1x1 convolution layer before applying a 3x3 kernel and increasing the number of channels with a 1x1 convolution layer. More information can be found in the original work [26].

A.2 Recurrent Neural Networks

A.2.1 Long Short-Term Memory

LSTMs contain gating mechanisms to control its two internal states. Cell internal states and output are described by the following equations:

$$f_t = \sigma(W^f x_t + U^f h_{t-1} + b^f) \tag{A.1}$$

$$\tilde{c}_t = \tanh\big(W^{\tilde{c}} x_t + U^{\tilde{c}} h_{t-1} + b^{\tilde{c}}\big) \tag{A.2}$$

$$i_t = \sigma(W^i x_t + U^i d_{t-1} + b_i) \tag{A.3}$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1} + b^o) \tag{A.4}$$

$$c_t = i_t \cdot \tilde{c}_t + f_t \odot c_{t-1} \tag{A.5}$$

$$h_t = o_t \odot \tanh(c_t) \tag{A.6}$$

$$y_t = h_t \tag{A.7}$$

where c_t and h_t are the hidden states and y_t is the output. f_t, $\tilde{c}_t$, i_t, and o_t are intermediary signals in the recurrent cell. σ denotes the sigmoid activation function and the superscripts of parameters W, U, and b differentiate between different network layers. The notation used is adopted from Mao [91].

A.2.2 Gated Recurrent Units

Similar to the LSTM, GRU cells contain gating mechanisms and one hidden state. The internal state and output are defined by the following equations:

$$r_t = \sigma(W^r x_t + U^r s_{t-1} + b^r) \tag{A.8}$$

$$\tilde{s}_t = \tanh\big(W^{\tilde{s}} x_t + U^{\tilde{s}}(r_t \odot s_{t-1}) + b^{\tilde{s}}\big) \tag{A.9}$$

$$z_t = \sigma(W^z x_t + U^z s_{t-1} + b^z) \tag{A.10}$$

$$s_t = z_t \odot \tilde{s}_t + (1 - z_t) \odot s_{t-1} \tag{A.11}$$

$$y_t = s_t \tag{A.12}$$

where r_t and z_t are internal cell signals and s_t and y_t are the hidden state and output respectively. σ denotes the sigmoid activation function and the superscripts of parameters W, U, and b differentiate between different network layers. The notation used is adopted from Mao [91].

Appendix B: Reinforcement Learning Algorithms

B.1 Model-Free

Algorithm 2.1 SARSA adapted from [45]

Initialize:
  step size $\alpha \in (0, 1]$, small $\epsilon > 0$
  $Q(s, a) \forall s \in S, a \in A$
while True **do**
  Initialize S
  Choose A given S from ϵ-greedy policy derived from Q
  while episode not terminated **do**:
    Take action A, observe R, S'
    Choose A' given S' from ϵ-greedy policy derived from Q
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'; A \leftarrow A';$
  end while
end while

Algorithm 2.2 Q-learning adapted from [45]

Initialize:
  step size $\alpha \in (0, 1]$, small $\epsilon > 0$
  $Q(s, a) \forall s \in S, a \in A$
while True **do**
  Initialize S
  while episode not terminated **do**:
    Choose A given S from ϵ-greedy policy derived from Q
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{A'} Q(S', A') - Q(S, A)\right]$
    $S \leftarrow S'$
  end while
end while

Algorithm 2.3 Proximal policy optimization adapted from [8]

Initialize:
  step size $\alpha \in (0, 1]$, small $\epsilon > 0$

while iteration $<$ N **do**
  while steps $<$ T **do**:
    Sample action from $\pi_{\theta_{old}}$, observe state s and reward r
    Compute advantage from $V_\phi(s)$ (via GAE [52])
  end while
  SGD step for K epochs on loss $\mathcal{J}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi(s_t,a_t,\theta)}{\pi(s_t,a_t,\theta_{old})} A_t, clip\left(\frac{\pi(s_t,a_t,\theta)}{\pi(s_t,a_t,\theta_{old})}, 1-\epsilon, 1+\epsilon\right) A_t\right)\right]$
  SGD step on loss $\mathcal{L}(\Phi) = \mathbb{E}[(V_\Phi(s_t) - \sum_t^T r_t)^2]$
end while