NAVIGATING TO OBJECTS: SIMULATION, DATA, AND MODELS

A Dissertation
Presented to
The Academic Faculty

By

Ram Ramrakhya

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
College of Computing
Department of Computer Science

Georgia Institute of Technology

May 2023

© Ram Ramrakhya 2023
I am built upon the small things I do every day, and the end results are no more than a byproduct of that.

*Kita Shinsuke*
ACKNOWLEDGMENTS

I have enjoyed every single day as an MS student at CVMLP lab and I have a bunch of people to thank for this awesome experience.

First, Dhruv Batra, for giving me the opportunity to intern with the lab when I did not have any background in ML research or Embodied AI. I was only good at solving the problem at hand using my engineering skills, Dhruv has taught me how to leverage those skills towards executing open research problems. He has taught me how to do research and how to critically think about important and impactful open problems. I would also like to thank him for guiding me on how to approach grad school applications, I have benefited a lot from his guidance through this process. I appreciate the chance to learn from him, and I would like to thank him for his constant support and encouragement.

I am grateful to have had the opportunity to work with Abhishek Das during my Master’s. Abhishek’s clarity of thought, ability to come up with important problems and communicate them, insightful feedback on ideas, experiments, and writing, critical feedback on slides and aesthetics, and preparing talks have taught me how to think, execute, interpret, and present research work.

I would also like to thank Devi Parikh. Devi’s attention to detail, constructive feedback, emphasis on the ability to clearly communicate ideas, and her time management system have shaped how I organize my thoughts to communicate clearly and manage my time effectively.

I am grateful to have had the opportunity to work with her.

My special thanks to Gunjan Aggarwal - for being a wonderful friend, for introducing me to the amazing experiences that you gain by exploring the unknown, and for being my go-to person for everything. I would like to thank her for teaching me how to enjoy life even during the most stressful times and for making my Master’s life super fun and better in every aspect.

I would like to thank each and every member of the Computer Vision Machine Learning
& Perception (CVMLP) Lab for providing a great culture and constructive input during my Masters. This includes - Erik Wijmans, Harsh Agrawal, Andrew Szot, Samyak Datta, Nirbhay Modhe, Meera Hahn, Joanne Truong, Naoki Yokoyama, Karmesh Yadav, Arjun Majumdar, Gunjan Aggarwal, Harsh Maheshwari, and Mukul Khanna. I feel incredibly privileged to have been a member of CVMLP Group.

I have been fortunate to have had the chance to collaborate with folks at FAIR, SFU and AI2 during the course of my Masters. I would particularly like to thank Eric Undersander for his suggestions on how to build scalable systems. I would also like to thank the OVRL, and HM3D-Semantics team - Karmesh Yadav, Oleksandr Maksymets, Manolis Savva, Devendra Singh Chaplot, Angel Xuan Chang, Santhosh Ramkrishnan, Theo Gervet, Aaron Gokaslan, Alex Clegg, and John Turner. From AI2, I would like to thank Luca Weihs and Ani Kembhavi for many insightful discussions on how to build a curriculum for embodied tasks, for suggesting interesting approaches to debug our experiments, and for their constant support.

To my parents Kanhaiya Ramrakhya and Radha Ramrakhya, for their significant scarifies over the years to support me, and for providing unconditional love, support, and encouragement to pursue my passion. And to my siblings Suraj Ramrakhya and Vanshita Ramrakhya for all the fun conversations and their constant support and encouragement even during stressful times.
# TABLE OF CONTENTS

Acknowledgments ................................................................. iv

List of Tables ............................................................................... x

List of Figures ........................................................................... xii

Summary ..................................................................................... xv

Chapter 1: Introduction .............................................................. 1

1.1 Outline ................................................................................. 1

Chapter 2: Habitat-Web: Learning Embodied Object-Search Strategies from Human Demonstrations at Scale ........................................... 4

2.1 Introduction ........................................................................... 4

2.2 Related Work ......................................................................... 7

2.2.1 Embodied Demonstrations from Humans ......................... 7

2.2.2 Exploration ....................................................................... 7

2.3 Habitat-WebGL Infrastructure ............................................. 8

2.3.1 Interface .......................................................................... 8

2.3.2 Habitat simulator and PsiTurk ......................................... 10

2.4 Tasks and Datasets ............................................................. 10
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.1</td>
<td>ObjectGoal Navigation</td>
<td>11</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Object Rearrangement – PICK&amp;PLACE</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Imitation Learning from Demonstrations</td>
<td>15</td>
</tr>
<tr>
<td>2.5.1</td>
<td>OBJECTNAV</td>
<td>16</td>
</tr>
<tr>
<td>2.5.2</td>
<td>PICK&amp;PLACE</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>Experiments &amp; Results</td>
<td>17</td>
</tr>
<tr>
<td>2.6.1</td>
<td>OBJECTNAV</td>
<td>17</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Habitat ObjectNav Challenge Results</td>
<td>19</td>
</tr>
<tr>
<td>2.6.3</td>
<td>PICK&amp;PLACE</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>Characterizing Learned Behaviors</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>Conclusion</td>
<td>24</td>
</tr>
</tbody>
</table>

Chapter 3: PIRLNav: Pretraining with Imitation and RL Finetuning for OBJECTNAV

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Related Work</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>OBJECTNAV and Imitation Learning</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1</td>
<td>OBJECTNAV</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2</td>
<td>OBJECTNAV Demonstrations</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Imitation Learning from Demonstrations</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>RL Finetuning</td>
<td>32</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Setup</td>
<td>32</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Finetuning Methodology</td>
<td>33</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Results</td>
<td>35</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>A.7.2 Feasibility of Off-Policy RL finetuning</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>A.8 Prior work in Imitation Learning</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>A.9 Training Details</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>A.9.1 Behavior Cloning</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>A.9.2 Reinforcement Learning</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>A.9.3 RL Finetuning using VPT</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>A.9.4 RL Finetuning Ablations</td>
<td>72</td>
<td></td>
</tr>
</tbody>
</table>
LIST OF TABLES

2.1 Dataset statistics for human demonstrations vs. shortest paths for OBJECT-NAV and PICK&PLACE. Coverage metrics are computed on subset of 1000 episodes. ................................................................. 10

2.2 OBJECTNAV results on MP3D-VAL. .................................................. 18

2.3 ObjectNav ablation results on the MP3D VAL split [2, 33]. ................. 19

2.4 Results on Habitat ObjectNav Challenge TEST-STD [52]. .................. 20

2.5 Pick-and-place results on splits constructed with unseen initializations in seen environments (1-3), with unseen instructions (4-6), and with unseen environments (7-9). .................................................. 21

2.6 Quantifying semantic exploration behaviors for IL agents trained on shortest paths (row 1) and human demonstrations (row 2), the Red Rabbit RL agent [15] (row 3), and humans (row 4). ............................................. 23

3.1 Comparison with VPT on HM3D VAL [19, 63] ............................... 36

3.2 RL-finetuning ablations on HM3D VAL [19, 63] ............................... 36

3.3 Results on HM3D TEST-STANDARD and TEST-CHALLENGE [87, 63]. Unpublished works submitted only to the OBJECTNAV leaderboard have been grayed out. .................................................. 37

3.4 Performance on HM3D VAL with imitation learning on SP, FE, and HD demonstrations. The size of each demonstration dataset is picked such that total steps of experience is ∼12M. ............................................. 39

3.5 Results on SP-favoring and FE-Favoring splits. ..................................... 41

A.1 ObjectNav results on the Gibson VAL split. ....................................... 59
A.2  Hyperparameters used for Imitation Learning.  . . . . . . . . . . . . . . . . .  71
A.3  Hyperparameters used for RL finetuning.  . . . . . . . . . . . . . . . . . . . .  72
A.4  RL-finetuning ablations on HM3D val [19, 63]  . . . . . . . . . . . . . . . .  72
LIST OF FIGURES

2.1 a) Example OBJECTNAV 1) human demonstration, 2) agent trained on human demonstrations, and 3) shortest path. Notice how humans demonstrate sophisticated exploration behavior to succeed at this task in unseen environments, which is hard to engineer into the right reward for an RL agent and is unlikely to be captured in shortest path demonstrations. An agent trained on human demonstrations learns this exploration and object-search behavior. b) Success on the OBJECTNAV MP3D-VAL split vs. no. of human demonstrations for training. ................................. 5

2.2 Screenshot of our Amazon Mechanical Turk interface for collecting OBJECTNAV demonstrations. Users are provided the agent’s first-person view of the environment and an instruction such as “Find and go to chair”. They can make the agent look around and move in the environment via keyboard controls, and can submit the task upon successful navigation by clicking ‘Submit’. .......................................................... 9

2.3 (a) Dataset statistics for human demonstrations vs. shortest paths for OBJECTNAV and PICK&PLACE. Coverage metrics are computed on subset of 1000 episodes. (b) Comparison of episode lengths and action histograms for human demonstrations vs. shortest paths. Human demonstrations are longer and have a more uniform action distribution than shortest paths. ............. 11

2.4 Our policy architectures for a) OBJECTNAV and b) PICK&PLACE. Both are simple CNN+RNN networks that embed and concatenate all sensory inputs, which are then fed into a GRU to predict actions. c) OBJECTNAV results on the MP3D VAL split [2, 33]. ................................. 15

2.5 Comparing RL and IL on (a) VAL success vs. no. of training steps, and (b) VAL success vs. no. of unique training steps. This distinguishes between an IL agent that learns from a static dataset vs. an RL agent that gathers unique trajectories on-the-fly. .................................................... 21

2.6 Visualizations of different learnt agent behaviors. Best viewed in the video at ram81.github.io/projects/habitat-web. ................................. 22
3.1 OBJECTNAV success rates of agents trained using behavior cloning (BC) vs. BC-pretraining followed by reinforcement learning (RL) (in blue). RL from scratch (i.e., BC=0) fails to get off-the-ground. With more BC demonstrations, BC success increases, and it transfers to even higher RL-finetuning success. But the difference between RL-finetuning vs. BC-pretraining success (in orange) plateaus and starts to decrease beyond a certain point, indicating diminishing returns with each additional BC demonstration.

3.2 OBJECTNAV trajectories for policies trained with BC→RL on 1) Human Demonstrations, 2) Shortest Paths, and 3) Frontier Exploration Demonstrations.

3.3 Learning rate schedule for RL Finetuning.

3.4 OBJECTNAV performance on HM3D VAL with BC-pretraining on shortest path (SP), frontier exploration (FE), and human demonstrations (HD), followed by RL-finetuning from each.

3.5 BC and RL performance for shortest paths (SP), frontier exploration (FE), and human demonstrations (HD) with equal BC training success on HM3D TRAIN (left) and VAL (right).

3.6 Success on ObjectNav HM3D VAL split vs. no. of frontier exploration demonstrations for training.

3.7 Failure modes of our best BC→RL OBJECTNAV policy.

A.1 Dataset size vs performance plot for PICK&PLACE.

A.2 Comparison of episode lengths and action histograms for human demonstrations vs. shortest paths for PICK&PLACE. Human demonstrations are longer and have a more uniform action distribution than shortest paths.

A.3 Per scene breakdown of GRTS for IL and RL agent on MP3D VAL split.

A.4 Visualizing multiple human demonstrations for OBJECTNAV all starting from the same start position and searching for ‘cabinet’.

A.5 a) Histogram of average SPL for each AMT user for OBJECTNAV. b) Plot showing average SPL for AMT users for OBJECTNAV. These plots clearly demonstrate some humans are better at solving the OBJECTNAV task than others.

xiii
A.6 Screenshot of our Amazon Mechanical Turk interface for collecting PICK\&PLACE demonstrations. Users are provided the agent’s first-person view of the environment and instruction such as "Pick the toy airplane and place it on the the colored wood blocks". They can make the agent look around and move in the environment via keyboard controls, and can submit the task upon successful completion by clicking the ‘Submit’ button. 63

A.7 Visualizations of learnt agent behaviors for OBJECTNAV and PICK\&PLACE. Best viewed in videos at sites.google.com/view/object-search-supp. 64

A.7 Visualizations of learnt agent behaviors for OBJECTNAV and PICK\&PLACE. Best viewed in videos at ram81.github.io/projects/habitat-web. 65

A.8 A policy pretrained on the OBJECTNAV task is used as initialization for actor weights and critic weights are initialized randomly for RL finetuning using DD-PPO. The policy performance immediately starts dropping early on during training and then recovers leading to slightly higher performance with further training. 73
SUMMARY

General-purpose robots that can perform a diverse set of embodied tasks in a diverse set of environments have to be good at visual exploration. Consider the canonical example of asking a household robot, ‘Where are my keys?’. To answer this (assuming the robot does not remember the answer from memory), the robot would have to search the house, often guided by intelligent priors – e.g. peeking into the washroom or kitchen might be sufficient to be reasonably sure the keys are not there, while exhaustively searching the living room might be much more important since keys are more likely to be there. While doing so, the robot has to internally keep track of where all it has been to avoid redundant search, and it might also have to interact with objects, e.g. check drawers and cabinets in the living room (but not those in the washroom or kitchen!).

This example illustrates fairly sophisticated exploration, involving a careful interplay of various implicit objectives (semantic priors, exhaustive search, efficient navigation, interaction, etc.) which are hard to learn using Reinforcement Learning (RL). In this thesis, we focus on learning such embodied object-search strategies from human demonstrations which implicitly captures intelligent behavior we wish to impart to our agents.

In Part I, we present a large-scale study of imitating human demonstrations on tasks that require a virtual robot to search for objects in new environments – (1) ObjectGoal Navigation (e.g. ‘find & go to a chair’) and (2) PICK&PLACE (e.g. ‘find mug, pick mug, find counter, place mug on counter’). In Part 2, we extend our focus to improving agents trained using human demonstrations in a tractable way. Towards this, we present PIRLNav, a two-stage learning scheme for BC pretraining on human demonstrations followed by RL-finetuning. Finally, using this BC→RL training recipe, we present a rigorous empirical analysis where we investigate whether human demonstrations can be replaced with ‘free’ (automatically generated) sources of demonstrations, e.g. shortest paths (SP) or task-agnostic frontier exploration (FE) trajectories.
CHAPTER 1
INTRODUCTION

1.1 Outline

General-purpose robots that can perform a diverse set of embodied tasks in a diverse set of environments have to be good at visual exploration. Consider the canonical example of asking a household robot, ‘Where are my keys?’ To answer this (assuming the robot does not remember the answer from memory), the robot would have to search the house, often guided by intelligent priors – e.g. peeking into the washroom or kitchen might be sufficient to be reasonably sure the keys are not there, while exhaustively searching the living room might be much more important since keys are more likely to be there. While doing so, the robot has to internally keep track of where all it has been to avoid redundant search, and it might also have to interact with objects, e.g. check drawers and cabinets in the living room (but not those in the washroom or kitchen!).

This example illustrates fairly sophisticated exploration, involving a careful interplay of various implicit objectives (semantic priors, exhaustive search, efficient navigation, interaction, etc.). Many recent tasks of interest in the embodied AI community – e.g. ObjectGoal Navigation [1, 2], rearrangement [3, 4], language-guided navigation [5, 6] and interaction [7], question answering [8, 9, 10, 11, 12] – involve some flavor of this visual exploration. With careful reward engineering, reinforcement learning (RL) approaches to these tasks have achieved commendable success [13, 14, 15, 16, 17]. However, engineering the ‘right’ reward function so that the learned policy exhibits desired behavior is unintuitive and frustrating (even for domain experts), expensive (requiring multiple rounds of retraining under different rewards), and not scalable to new tasks or behaviors. Modular learning (ML) methods for such tasks have also emerged as a strong competitor [17, 18, 19]. These methods rely on
separate modules for semantic mapping that build explicit structured map representations, a high-level semantic exploration module that is learned through RL to solve the ‘where to look?’ subproblem, and a low-level navigation policy that solves ‘how to navigate to \((x, y)\)?’.

In this thesis, we take a different perspective and advance the alternative agenda of learning visual exploration from human demonstrations which implicitly captures intelligent behavior we wish to impart to our agents using simple end-to-end trained neural networks.

In Chapter 2, we present a large-scale study of imitating human demonstrations on tasks that require a virtual robot to search for objects in new environments – (1) ObjectGoal Navigation (e.g. ‘find & go to a chair’) and (2) PICK&PLACE (e.g. ‘find mug, pick mug, find counter, place mug on counter’). To do so, we first develop a virtual teleoperation data-collection infrastructure (Habitat-Web) – connecting Habitat simulator running in a web browser to Amazon Mechanical Turk, allowing remote users to teleoperate virtual robots, safely and at scale. We collect one of the largest dataset of human demonstrations in simulation for 2 embodied tasks using Habitat-Web. Next, we use this data to answer the question – how does large-scale imitation learning (IL) (which has not been hitherto possible) compare to reinforcement learning (RL) (which is the status quo) or modular learning (ML) methods? On both tasks, we find that IL (with no bells or whistles) using human demonstrations outperforms RL and ML methods. We also find the IL-trained agent learns efficient object-search behavior from humans – it peeks into rooms, checks corners for small objects, turns in place to get a panoramic view – none of these are exhibited as prominently by the RL or ML agents, and to induce these behaviors via contemporary RL techniques would require tedious reward engineering or hardcoded heuristics.

In Chapter 3, we extend our focus to improving agents trained using human demonstrations in a tractable way. Towards this, we present PIRLNav, a two-stage learning scheme for BC pretraining on human demonstrations followed by RL-finetuning. Using this BC→RL training recipe, we present a rigorous empirical analysis of design choices. First, we investi-
gate whether human demonstrations can be replaced with ‘free’ (automatically generated) sources of demonstrations, e.g. shortest paths (SP) or task-agnostic frontier exploration (FE) trajectories. We find that BC→RL on human demonstrations outperforms BC→RL on SP and FE trajectories, even when controlled for the same BC-pretraining success on TRAIN, and even on a subset of VAL episodes where BC-pretraining success favors the SP or FE policies. Next, we study how RL-finetuning performance scales with the size of the BC pretraining dataset. We find that as we increase the size of the BC-pretraining dataset and get to high BC accuracies, the improvements from RL-finetuning are smaller, and that 90% of the performance of our best BC→RL policy can be achieved with less than half the number of BC demonstrations. Finally, we analyze failure modes of our OBJECTNAV policies, and present guidelines for further improving them.
CHAPTER 2
HABITAT-WEB: LEARNING EMBODIED OBJECT-SEARCH STRATEGIES FROM HUMAN DEMONSTRATIONS AT SCALE

2.1 Introduction

General-purpose robots that can perform a diverse set of embodied tasks in a diverse set of environments have to be good at visual exploration. Consider the canonical example of asking a household robot, ‘Where are my keys?’. To answer this (assuming the robot does not remember the answer from memory), the robot would have to search the house, often guided by intelligent priors – e.g. peeking into the washroom or kitchen might be sufficient to be reasonably sure the keys are not there, while exhaustively searching the living room might be much more important since keys are more likely to be there. While doing so, the robot has to internally keep track of where all it has been to avoid redundant search, and it might also have to interact with objects, e.g. check drawers and cabinets in the living room (but not those in the washroom or kitchen!).

This example illustrates fairly sophisticated exploration, involving a careful interplay of various implicit objectives (semantic priors, exhaustive search, efficient navigation, interaction, etc.). Many recent tasks of interest in the embodied AI community – e.g. ObjectGoal Navigation [1, 2], rearrangement [3, 4], language-guided navigation [5, 6] and interaction [7], question answering [8, 9, 10, 11, 12] – involve some flavor of this visual exploration. With careful reward engineering, reinforcement learning (RL) approaches to these tasks have achieved commendable success [13, 14, 15, 16, 17]. However, engineering the ‘right’ reward function so that the learned policy exhibits desired behavior is unintuitive and frustrating (even for domain experts), expensive (requiring multiple rounds of retraining under different rewards), and not scalable to new tasks or behaviors. For complex tasks (e.g. object rear-
Figure 2.1. a) Example OBJECTNAV 1) human demonstration, 2) agent trained on human demonstrations, and 3) shortest path. Notice how humans demonstrate sophisticated exploration behavior to succeed at this task in unseen environments, which is hard to engineer into the right reward for an RL agent and is unlikely to be captured in shortest path demonstrations. An agent trained on human demonstrations learns this exploration and object-search behavior. b) Success on the OBJECTNAV MP3D-VAL split vs. no. of human demonstrations for training.

In this work, we advance the alternative research agenda of imitation learning [20] – i.e., collecting a large dataset of human demonstrations (that implicitly capture intelligent behavior we wish to impart to our agents) and learning policies directly from these human demonstrations.

First, we develop a safe scalable virtual teleoperation data-collection infrastructure – connecting the Habitat simulator running in a browser to Amazon Mechanical Turk (AMT). We develop this in a way that enables collecting human demonstrations for a variety of tasks being studied within the Habitat [21, 22] ecosystem (e.g. PointNav [2], OBJECTNAV [1, 2], ImageNav [23], VLN-CE [6], MultiON [24], etc.).

We use this infrastructure to collect human demonstration datasets for 2 tasks requiring visual search – 1) ObjectGoal Navigation (e.g. ‘find & go to a chair’) and 2) PICK&PLACE (e.g. ‘find mug, pick mug, find counter, place on counter’). In total we collect 92k human demonstrations, 80k demonstrations for OBJECTNAV and 12k demonstrations for PICK&PLACE. In contrast, the largest existing datasets have 3-10k human demonstrations in simulation [25, 26, 27] or on real robots [28, 29], an order of magnitude smaller. This virtual teleoperation data contains 29.3M actions, which is equivalent to 22,600 hours of
real-world teleoperation time assuming a LoCoBot motion model from [30] (details in appendix (Sec. A.3)). The first thing this data provides is a ‘human baseline’ with sufficiently tight error-bars to be taken seriously. On the OBJECTNAV validation split, humans achieve 93.7±0.1% success and 42.5±0.5% Success Weighted by Path Length (SPL) [2] (vs. 34.6% success and 7.9% SPL for the 2021 Habitat ObjectNav Challenge winner [15]). The success rate (93.7%) suggests that this task is largely doable for humans (but not 100%). The SPL (42.5%) suggests that even humans need to explore significantly.

Beyond scale, the data is also rich and diverse in the strategies that humans use to solve the tasks. Fig. 2.1 shows an example trajectory of an AMT user controlling a LoCoBot looking for a ‘plant’ in a new house – notice the peeking into rooms, looping around the dining table – all of which is (understandably) absent from the shortest path to the goal.

We use this data to answer the question – how does large-scale imitation learning (IL) (which has not been hitherto possible) compare to large-scale reinforcement learning (RL) (which is the status quo)? On OBJECTNAV, we find that IL (with no bells or whistles) using only 70$k human demonstrations outperforms RL using 240$k agent-gathered trajectories. This effectively establishes an ‘exchange rate’ – a single human demonstration appears to be worth ~4 agent-gathered ones. More importantly, we find the IL-trained agent learns efficient object-search behavior – as shown in Fig. 2.1 and Sec. 2.7. The IL agent learns to mimic human behavior of peeking into rooms, checking corners for small objects, turning in place to get a panoramic view – none of these are exhibited as prominently by the RL agent. Finally, the accuracy vs. training-data-size plot (Fig. 2.1b) shows promising scaling behavior, suggesting that simply collecting more demonstrations is likely to advance the state of art further. On PICK&PLACE, the comparison is even starker – IL-agents achieve ~18% success on episodes with new object-receptacle locations when trained with 9.5$k human demonstrations, while RL agents fail to get beyond 0%.

On both tasks, we find that demonstrations from humans are essential; imitating shortest paths from an oracle produces neither accuracy nor the strategic search behavior. In hindsight,
this is perfectly understandable – shortest paths (e.g. Fig. 2.1(a3)) do not contain any exploration but the task requires the agent to explore. Essentially, a shortest path is inimitable, but imitation learning is invaluable. Overall, our work provides compelling evidence for investing in large-scale imitation learning of human demonstrations.

2.2 Related Work

2.2.1 Embodied Demonstrations from Humans

Prior expert demonstration datasets for embodied tasks combining vision and action (and optionally language) can be broadly categorized into either consisting of shortest-path trajectories from a planner with privileged information [7, 8, 31, 5], or consisting of human-provided trajectories [26, 27, 25]. While some works in the former collect natural language data from humans [7, 5], we contend that collecting navigation data from humans is equally crucial. Datasets with human-provided navigation trajectories are typically small. TEACh [25], CVDN [26] and WAY [27] have <10k episodes, while the EmbodiedQA [8] dataset has ∼700 human-provided episodes – all prohibitively small for training proficient agents. A key contribution of our work is a scalable web-based infrastructure for collecting human navigation and interaction demonstrations, that is easily extensible to any task situated in the Habitat [21] simulator, including language-based tasks. We have collected ∼13x more demonstrations (in total 92k) compared to prior publicly available datasets. In a similar vein, Abramson et al. [32] study large-scale imitation learning on ∼600k human demonstrations, but their dataset is not publicly available and the environments used lack visual realism compared to Matterport3D [33].

2.2.2 Exploration

Learning how to explore an environment to gather sufficient information for use in downstream tasks has a rich history [34]. Curiosity-based approaches typically use reinforcement learning to maximize intrinsic rewards that capture the surprise or state prediction error of
the agent [35, 36, 37]. State visitation count rewards are also popular for learning exploration [38, 39]. We refer the reader to Ramakrishnan et al. [19] for a review of exploration objectives for embodied agents. For improving exploration in OBJECTNAV specifically, SemExp [17] made use of a modular policy for semantic mapping and path planning, Ye et al. [15] used time-decaying state visitation count reward, and Maksymets et al. [16] used area coverage reward.

Most relatedly, Chen et al. [40] used $\sim 700$ human navigation trajectories from the EmbodiedQA dataset [8] (ignoring the questions) to learn task-independent exploration using imitation learning. We likewise train agents via imitation learning on human demonstrations, but rather than encouraging task-agnostic exploration, we consider human demonstrations to be a rich *task-specific* mix of exploration and efficient navigation, that simple architectures without explicit mapping and planning modules can be trained on.

2.3 Habitat-WebGL Infrastructure

To be able to train agents via imitation learning on human demonstrations, we first need a reliable pipeline to collect human demonstrations at scale. To this end, we develop a web-based setup to connect the Habitat simulator [21, 22] to AMT users, building on the work of Newman et al. [41].

2.3.1 Interface

Fig. 2.2 shows a screenshot of the interface an AMT user interacts with to complete a data collection task. This web application renders assets from Habitat-Sim running on the user’s browser via WebGL. All data collection in this work was done in Matterport3D [33] scans, but any Habitat-compatible asset may be used in future. Users can see the agent’s first-person RGB view, and can move around and grab / release objects using keyboard controls. On the task page, users are provided an instruction and details about keyboard controls to complete the task. For OBJECTNAV, we provide an instruction of the form ‘Find
Figure 2.2. Screenshot of our Amazon Mechanical Turk interface for collecting OBJECTNAV demonstrations. Users are provided the agent’s first-person view of the environment and an instruction such as “Find and go to chair”. They can make the agent look around and move in the environment via keyboard controls, and can submit the task upon successful navigation by clicking ‘Submit’.

_and go to the <goal_object_category>’. For tasks requiring interaction with objects (e.g. PICK&PLACE), we highlight the object under the user’s gaze by drawing a 3D bounding box around it (pointed to by a crosshair as in video games). In our initial pilots, we found this to improve user experience when grabbing objects instead of users having to guess when objects are available to be picked up. When an object is successfully grabbed, it disappears from the first-person view and immediately appears in the ‘inventory’ area on the task interface. When a grabbed object is released, it is dropped at the center of the user’s screen where the crosshair would be pointing to. If the crosshair points to a distance, the object is dropped on the floor from a height at a distance of 1m from the agent’s location.
Table 2.1. Dataset statistics for human demonstrations vs. shortest paths for OBJECTNAV and PICK&PLACE. Coverage metrics are computed on subset of 1000 episodes.

<table>
<thead>
<tr>
<th></th>
<th>OBJECTNAV</th>
<th></th>
<th>PICK-AND-PLACE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Human</td>
<td>Shortest Path</td>
<td>Human</td>
<td>Shortest Path</td>
</tr>
<tr>
<td>1) Total Episodes</td>
<td>80,217</td>
<td>114,165</td>
<td>11,955</td>
<td>25,747</td>
</tr>
<tr>
<td>2) Success</td>
<td>88.9%</td>
<td>100.0%</td>
<td>86.3%</td>
<td>100.0%</td>
</tr>
<tr>
<td>3) SPL</td>
<td>39.9%</td>
<td>94.9%</td>
<td>21.2%</td>
<td>90.9%</td>
</tr>
<tr>
<td>4) Occupancy coverage</td>
<td>17.9%</td>
<td>4.6%</td>
<td>26.5%</td>
<td>9.2%</td>
</tr>
<tr>
<td>5) Sight coverage</td>
<td>67.7%</td>
<td>33.2%</td>
<td>70.3%</td>
<td>42.5%</td>
</tr>
</tbody>
</table>

Upon completion, users submit the task by clicking ‘Submit’. At this point, the sequence of keyboard actions, agent, and object states are recorded in our backend server.

2.3.2 Habitat simulator and PsiTurk

Our Habitat-WebGL application is developed in Javascript, and allows us to access all C++ simulator APIs through Javascript bindings. This lets us use the full set of simulation features available in Habitat. To simulate physics, we use the physics APIs from Habitat 2.0 [22], including rigid body dynamics support (C++ APIs exposed as Javascript bindings). Our interface executes actions entered by users every 50ms (rendering 20 frames per second) and then steps physics for 50ms in the simulator. All of our tasks on AMT are served using PsiTurk and an NGINX reverse proxy, and all data stored in a MySQL database. We use PsiTurk to manage the tasks as it provides us with useful helper functions to log task-related metadata, as well as launch and approve tasks.

See Sec. A.6 for details on how we validate human-submitted AMT tasks and ensure data quality.

2.4 Tasks and Datasets

Using our web infrastructure, we collect demonstration datasets for two embodied tasks – OBJECTNAV [1, 2] and PICK&PLACE, an instantiation of object rearrangement [3].
2.4.1 ObjectGoal Navigation

In the ObjectGoal Navigation (OBJECTNAV) task, an agent is tasked with navigating to an instance of a specified object category (e.g., ‘chair’) in an unseen environment. The agent does not have access to a map of the environment and must navigate using an RGBD camera and a GPS+Compass sensor which provides location and orientation information relative to the start of the episode. The agent also receives the goal object category ID as input. The full action space is discrete and consists of MOVE_FORWARD (0.25m), TURN_LEFT (30°), TURN_RIGHT (30°), LOOK_UP (30°), LOOK_DOWN (30°), and STOP actions. For the episode to be considered successful, the agent must stop within 1m Euclidean distance of the goal object within a maximum of 500 steps and be able to turn to view the object from that end position [42].

Human Demonstrations (OBJECTNAV-HD)

We collect 70k demonstrations on the 56 training scenes from Matterport3D [33] following the standard splits defined in [33, 2]. For each scene, we collect ~59 demonstration episodes for each unique goal object category with a randomly set start location of the human demonstrator for each episode. This amounts to an average of ~1250 demonstrations per scene. Additionally, we collect 10k demonstrations on 25 training scenes from Gibson [43]. For each Gibson scene, we collect ~66 demonstration episodes for each unique goal object category. This amounts to ~396 demonstrations per scene. Similar to when training artificial
agents, humans can view first-person RGB on the task interface, but unlike artificial agents, humans do not get access to Depth and GPS+Compas. We assume humans are sufficiently proficient at inferring depth and odometry from vision, to the extent required to accomplish the goal. In total, we collect $80k$ OBJECTNAV demonstrations amounting to $\sim19.5M$ steps of experience, each episode averaging 243 steps.

Shortest Path Demonstrations

To compare against prior embodied datasets of shortest paths [7, 8, 31, 5] and to demonstrate the unique advantage of human demonstrations, we also generate a dataset of shortest paths. The analysis in this section was performed on a subset of 35k demonstrations of OBJECTNAV-HD (collected in first phase). These demonstrations are generated by greedily fitting actions to follow the geodesic shortest path to the nearest navigable goal object viewpoint. Since shortest paths are (by design) shorter than human demonstrations (average 67 vs. 243 steps per demonstration), we compensate by generating a larger number of shortest paths to roughly match the steps with 35k human demonstrations (7.6M steps from 114k shortest paths vs. 8.4M steps from 35k human demonstrations).

Analysis

Tab. 2.1 reports statistics of our human and shortest path demonstration datasets. Recall that an episode is considered a failure if the target object is not found within 500 navigation steps. Under this definition, humans fail on 11.1% training set episodes; they fail on 0% episodes if we relax the step-limit to the maximum number of actions humans took i.e. 2000 steps. Surprisingly, SPL for humans is 39.9% for training split episodes, significantly lower than 94.9% for shortest paths underscoring the difficulty in searching for objects in in unseen environments.

We additionally report two metrics to demonstrate that the OBJECTNAV task requires significant exploration. Occupancy Coverage (OC) measures percentage of total area
covered by the agent when navigating. To compute OC, we first divide the map into voxel grids of $2.5m \times 2.5m \times 2.5m$ and increment a counter for each visited voxel. Sight Coverage (SC) measures the percentage of total navigable area visible to the agent in its field of view (FOV) during an episode. To compute SC, we project a mask on the top-down map of the environment using the agent’s FOV, that is iteratively updated at every step to update the area seen by the agent. OC and SC metrics for human demonstrations show that humans traverse 3-4x and observe 2x the area of the environment when performing this task compared to shortest paths.

Fig. 2.3a,b show episode length and action histograms for human and shortest path demonstrations. Human demonstrations are longer (average $\sim 243$ vs. $\sim 67$ steps per demonstration) and have a slightly more uniform action distribution.

2.4.2 Object Rearrangement – PICK&PLACE

In the pick-and-place task (PICK&PLACE), an agent must follow an instruction of the form ‘Place the <object> on the <receptacle>’, without being told the location of the <object> or <receptacle> in a new environment. The agent must explore and navigate to the object, pick it up, explore and navigate to the receptacle, and place the previously picked-up object on it. Similar to OBJECTNAV, agents are not equipped with a map of the environment, and only have access to an RGBD camera and a GPS+compass sensor. At a high level, PICK&PLACE can be thought of as a natural extension of OBJECTNAV, performing it twice in the same episode – once to find the specified object and again to find the specified receptacle – delimited by grab and release actions. For object interaction, we use the ‘magic pointer’ abstraction defined in [3]. If the agent is not holding any object, the grab/release action will pick the object pointed to by its crosshair (at the center of its viewpoint) if within $1.5m$ of the object. If the agent is already holding an object, the grab/release action will drop the object at the crosshair location. If there is no drop-off point within $1.5m$ in the direction of the crosshair, the object will be dropped on the floor $1m$ in front of the agent. The full action
space is discrete and consists of MOVE_FORWARD (0.15m), MOVE_BACKWARD (0.15m), 
TURN_LEFT (5°), TURN_RIGHT (5°), LOOK_UP (5°), LOOK_DOWN (5°), GRAB_RELEASE, 
NO_OP (step physics 50ms), and STOP. For the episode to be considered successful, the 
agent must place the object on top of the receptacle – i.e. the object center should be at a 
height greater than the receptacle center, and within 0.7m of the receptacle object center – 
within 1500 steps. We picked this 0.7m threshold distance between the object and receptacle 
based on pilots on AMT. 0.7m was sufficiently strict for avoiding false positives in the 
collected demonstrations where users are able to submit the task without necessarily placing 
the object on top of the receptacle.

**Human Demonstrations (PICK&PLACE-HD)**

We collect human demonstrations for PICK&PLACE on 9 scenes from Matterport3D [33]. 
In each episode, objects and receptacles are instantiated by randomly sampling from 457 
possible object-receptacle pairs. We initialize the object and receptacle at randomly sampled 
locations in the environment, and collect 3 demonstrations for each object-receptacle pair. 
The agent, object, and receptacle locations are randomized across all episodes (including 
the 3 we collect for each object-receptacle pair). In total, we have $457 \times 3$ unique object-
receptacle-agent position initializations per scene, amounting to $457 \times 3 \times 9 = \sim 12k$ 
demonstrations, which is $\sim 11.5M$ steps in experience, each episode averaging 932 steps.

**Shortest Path Demonstrations**

Similar to OBJECTNAV, we generate shortest path demonstrations for PICK&PLACE. These 
demonstrations are generated by first using the geodesic shortest-path follower to the object, 
then using a heuristic action planner to face and pick up the object, then following the 
geodesic shortest-path to the receptacle, and again using a heuristic action planner to 
drop the object on the receptacle. We generated 25.7k shortest path demonstrations for 
PICK&PLACE, each averaging 342 steps, amounting to a total of $\sim 8.8$ million steps of
Figure 2.4. Our policy architectures for a) OBJECTNAV and b) PICK&PLACE. Both are simple CNN+RNN networks that embed and concatenate all sensory inputs, which are then fed into a GRU to predict actions. c) OBJECTNAV results on the MP3D VAL split [2, 33].

Analysis

Tab. 2.1 reports statistics for human and shortest path demonstrations. Similar to OBJECTNAV, humans have significantly lower SPL, and 2x higher occupancy and sight coverage compared to shortest paths, suggesting the need for exploration. Comparing episode lengths and action histograms (see appendix ( Sec. A.1.1) for figure), human demonstrations are longer and make use of all 9 actions. Interestingly, humans often use the MOVE_BACKWARD action to backtrack, which the shortest path agents do not use (by design), instead of turning 180° and moving forward. This behavior does not appear in OBJECTNAV shortest path demonstrations because there is just one target object, and so the geodesic shortest path would never involve backtracking or making 180° turns.

2.5 Imitation Learning from Demonstrations

We use behavior cloning to learn a policy from demonstrations. Let \( \pi_\theta(a_t | o_t) \) denote a policy parametrized by \( \theta \) that maps observations \( o_t \) to a distribution over actions \( a_t \). Let \( \tau \) denote a trajectory consisting of state, observation, human action tuples: \( \tau = (s_0, o_0, a_0, \ldots, s_T, o_T, a_T) \) and \( \mathcal{T} = \{ \tau^{(i)} \}_{i=1}^N \) denote a dataset of human demonstrations.
The learning problem can be summarized as:

$$
\theta^* = \arg\min_\theta \sum_{i=1}^N \sum_{(o_t, a_t) \in \tau(i)} - \log \left( \pi_\theta(a_t|o_t) \right)
$$

(2.1)

**Inflection weighting** introduced in Wijmans et al. [9], adjusts the loss function to upweight timesteps where actions change (i.e. \(a_{t-1} \neq a_t\)). Specifically, the inflection weighting loss coefficient is computed as total no. of actions in the dataset divided by the total no. of inflection points, and this coefficient is multiplied with the loss at each inflection timestep where \(a_{t-1} \neq a_t\). This approach was found to be useful for tasks like navigation with long sequences of the same actions, e.g. several ‘forward’ actions when navigating corridors [9]. We use inflection weighting in all our experiments and found it to help over vanilla behavior cloning.

Our **base policy** is a simple CNN+RNN architecture. We first embed all sensory inputs using feed-forward modules. For RGB, we use a randomly initialized ResNet18 [44]. For depth, we use a ResNet50 that was pretrained on PointGoal navigation using DD-PPO [13]. Then these RGB and depth features (and optionally other task-specific features) are concatenated and fed into a GRU [45] to predict a distribution over actions \(a_{t+1}\). Task-specific architectural choices over this base policy are described in the next sections.

### 2.5.1 OBJECTNav

Fig. 2.4a shows our OBJECTNav architecture. Similar to Anand et al. [46], we feed in RGBD inputs of size 640 \(\times\) 480 passed through a 2X2-AVGPOOL layer to reduce the resolution (performing low-pass filtering + downsampling). The agent also has a GPS+Compass sensor, which provides location and orientation relative to start of the episode. GPS+Compass inputs are pass through fully-connected layers to embed them to 32-d vectors. In addition to RGBD and GPS+Compass, following Ye et al. [15], we use two additional semantic features – semantic segmentation (SemSeg) of the input RGB and a ‘Semantic Goal Exists’ (SGE) scalar which is the fraction of the visual input occupied by the goal category. These semantic
features are computed using a pretrained and frozen RedNet [47] that was pretrained on SUN RGB-D [48] and finetuned on 100k randomly sampled front-facing views rendered in the Habitat simulator. Finally, we also feed in the object goal category embedded into a 32-d vector. All of these input features are concatenated to form an observation embedding, and fed into a 2-layer, 512-d GRU at every timestep. We train this policy for ∼400M steps (= ∼21 epochs on ∼70k demonstration episodes). We evaluate checkpoints at every ∼15M steps for the last 50M steps of training, and report metrics for checkpoints with the highest success on the validation split.

2.5.2 PICK&PLACE

Fig. 2.4b shows our PICK&PLACE architecture. We feed in RGBD inputs of size 256 × 256. In addition to RGBD observations, the policy gets as input language instructions of the form ‘Place the <object> on the <receptacle>’ encoded using a single-layer LSTM [49]. RGBD and instruction features are concatenated to form an observation embedding, which is fed into a 2-layer, 512-d GRU at every timestep. We train this policy for ∼90M steps (= ∼10 epochs on ∼9.5k demonstration episodes). We evaluate checkpoints at every ∼10M steps during training, and report metrics for checkpoints with the highest success on the validation split.

2.6 Experiments & Results

2.6.1 OBJECTNav

Tab. 2.2 reports results on the MP3D VAL split for several baselines. First, we compare our approach with two state-of-the-art RL approaches from prior work. Maksymets et al. [16] (row 1) train their policy using a reward structure that breaks OBJECTNav into two subtasks – exploration and direct navigation to goal object once it is spotted. This agent gets a positive reward for maximizing area coverage until it sees the goal object. It then receives a navigation reward to minimize distance-to-object. This policy achieves 20.0%
Table 2.2. OBJECTNAV results on MP3D-VAL.

<table>
<thead>
<tr>
<th>Method</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) RL (ExploreTillSeen) [16]</td>
<td>20.0%</td>
<td>6.5%</td>
</tr>
<tr>
<td>2) RL (ExploreTillSeen + THDA) [16]</td>
<td>28.4%</td>
<td>11.0%</td>
</tr>
<tr>
<td>3) RL (Red Rabbit) [15]</td>
<td>34.6%</td>
<td>7.9%</td>
</tr>
<tr>
<td>4) RL (EmbCLIP) [50]</td>
<td>21.6%</td>
<td>8.7%</td>
</tr>
<tr>
<td>5) IL w/ Shortest Paths</td>
<td>4.4%</td>
<td>2.2%</td>
</tr>
<tr>
<td>6) IL w/ 35k Human Demos (similar #steps as row 4)</td>
<td>31.6%</td>
<td>8.5%</td>
</tr>
<tr>
<td>7) IL w/ 50k Human Demos</td>
<td>32.4%</td>
<td>9.1%</td>
</tr>
<tr>
<td>8) IL w/ 50k Human Demos (includes 10k THDA [16])</td>
<td>33.2%</td>
<td>9.5%</td>
</tr>
<tr>
<td>9) IL w/ 70k Human Demos</td>
<td>35.4%</td>
<td>10.2%</td>
</tr>
<tr>
<td>10) IL w/ 70k Human Demos (includes 10k Gibson [43])</td>
<td>33.9%</td>
<td>9.7%</td>
</tr>
<tr>
<td>11) IL w/ 80k Human Demos</td>
<td>33.8%</td>
<td>9.9%</td>
</tr>
<tr>
<td>12) Humans</td>
<td>93.7%</td>
<td>42.5%</td>
</tr>
</tbody>
</table>

success and 6.5% SPL (row 1). [16] then combine this reward structure with Treasure Hunt Data Augmentation (THDA) – inserting arbitrary 3D target objects in the scene to augment the set of training episodes. With THDA, this achieves 28.4% success and 11.0% SPL (row 2), 7.0% worse and 0.8% better respectively than behavior cloning on 70k human demonstrations (row 9). Ye et al. [15] (row 3) train their policy with a combination of exploration and distance-based navigation rewards, and their representations with several auxiliary tasks (e.g. inverse dynamics and predicting map coverage). This achieves 34.6% success and 7.9% SPL (row 3), which is 0.8% worse on success and 2.3% worse on SPL than our approach (row 9). Khandelwal et al. [50] (row 4) train a policy using CLIP [51] as a visual backbone with simple distance-based navigation rewards. This achieves 21.6% success and 8.7% SPL (row 4), which is 13.8% worse on success and 1.5% worse on SPL than our approach (row 9). IL on a dataset of shortest paths achieves 4.4% success and 2.2% SPL (row 5), significantly worse than training on 35k human demonstrations (31.6% success, 8.5% SPL). Recall that comparison of shortest path demonstrations was done with a subset of 35k OBJECTNAV-HD demonstrations that were collected in the first phase of the project. Next, we also collected 10k human demonstrations on OBJECTNAV episodes generated in THDA fashion – i.e. asking humans to find randomly inserted objects. Notice that this involves pure exhaustive search, since there are no semantic priors that humans
Table 2.3. ObjectNav ablation results on the MP3D VAL split [2, 33].

<table>
<thead>
<tr>
<th>Method</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) IL wo/ Vision</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>2) IL wo/ Semantic Input</td>
<td>22.7%</td>
<td>6.1%</td>
</tr>
<tr>
<td>3) IL w/ RGBD + Semantic Input</td>
<td>31.6%</td>
<td>8.5%</td>
</tr>
</tbody>
</table>

can leverage in this setting. An IL agent trained on 10k THDA demonstrations combined with the original 40k demonstrations achieves 33.2% success and 9.5% SPL (row 8) which is 0.8% better on success and 0.4% better on SPL than 50k non-THDA demonstrations (row 7), i.e. adding these THDA demonstrations with exhaustive search behavior helps. We also collected 10k demonstrations on Gibson[43] OBJECTNAV episodes to compare effect of different scene datasets. An agent trained on 10k Gibson demonstrations combined with 60k MP3D demonstrations achieves 33.9% success and 9.7% SPL (row 11), which is 1.5% worse on success and 0.5% worse on SPL compared to when we use MP3D-only demonstrations (row 9).

Finally, we also benchmark human performance on the MP3D VAL split – 93.7% success, 42.5% SPL (row 12).

ObjectNav Sensor Ablations

Tab. 2.3 reports results on the MP3D VAL split for various ablations of our approach trained on 35k human demonstrations. First, without any visual input (row 1), i.e. no RGBD and semantic inputs, the agent fails to learn anything (0% success, 0% SPL). Second, without SemSeg and SGE features (and keeping only RGB and Depth features) to the policy, performance drops by 8.9% success and 2.4% SPL (row 2 vs. 3).

2.6.2 Habitat ObjectNav Challenge Results

Tab. 2.4 compares our results with prior approaches from the 2020 and 2021 Habitat Challenge leaderboards. Our approach (IL w/ 70k demonstrations) achieves 27.8% success and 9.9% SPL (row 8), outperforming prior RL-trained counterparts – 3.3% better success,
3.5% better SPL than Red Rabbit (6-Act Base) [15] (row 5), and 6.7% better success, 1.1% better SPL than ExploreTillseen + THDA [16] (row 7).

Table 2.4. Results on Habitat ObjectNav Challenge TEST-STD [52].

<table>
<thead>
<tr>
<th>Team / Method</th>
<th>Success (+)</th>
<th>SPL (+)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) DD-PPO baseline [13, 15]</td>
<td>6.2%</td>
<td>2.1%</td>
</tr>
<tr>
<td>2) Active Exploration (Pre-explore)</td>
<td>8.9%</td>
<td>4.1%</td>
</tr>
<tr>
<td>3) SRCB-robot-sudoer</td>
<td>14.4%</td>
<td>7.5%</td>
</tr>
<tr>
<td>4) SemExp [53]</td>
<td>17.9%</td>
<td>7.1%</td>
</tr>
<tr>
<td>5) Red Rabbit (6-Act Base) [15]</td>
<td>24.5%</td>
<td>6.4%</td>
</tr>
<tr>
<td>6) Red Rabbit (6-Act Tether) [15]</td>
<td>21.1%</td>
<td>8.1%</td>
</tr>
<tr>
<td>7) ExploreTillSeen + THDA [16]</td>
<td>21.1%</td>
<td>8.8%</td>
</tr>
<tr>
<td>8) IL w/ 70k Human Demos</td>
<td>27.8%</td>
<td>9.9%</td>
</tr>
</tbody>
</table>

**Performance vs. Dataset size**

To investigate scaling behavior, we plot VAL success against the size of the human demonstrations dataset in Fig. 2.1b. We created splits of the human demonstrations’ dataset of increasing sizes, from 4k to 70k, and trained models with the same set of hyperparameters on each split. All hyperparameters were picked early in the course of the data collection (on the 4k and 12k subsplits) and fixed for later experiments. So VAL performance in the small-data regime may be an optimistic estimate and in the large data regime a pessimistic estimate. True scaling behavior may be even stronger. Increasing dataset size consistently improves performance and has not yet saturated, suggesting that simply collecting more demonstrations is likely to lead to further gains.

**Sample Efficiency**

Fig. 2.5 plots VAL success against no. of training steps of experience (in millions) in Fig. 2.5a and against unique steps of experience in Fig. 2.5b. Recall that IL involves ∼21 epochs on a static dataset of ∼70k demos, while RL (from [15]) gathers unique agent-driven trajectories on-the-fly. Fig. 2.5a shows that IL behaves like supervised learning (as expected) with
Figure 2.5. Comparing RL and IL on (a) VAL success vs. no. of training steps, and (b) VAL success vs. no. of unique training steps. This distinguishes between an IL agent that learns from a static dataset vs. an RL agent that gathers unique trajectories on-the-fly.

Table 2.5. Pick-and-place results on splits constructed with unseen initializations in seen environments (1-3), with unseen instructions (4-6), and with unseen environments (7-9).

<table>
<thead>
<tr>
<th>Method</th>
<th>VAL Success % (↑)</th>
<th>SPL % (↑)</th>
<th>TEST Success % (↑)</th>
<th>SPL % (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) IL w/ Shortest Paths</td>
<td>1.9</td>
<td>1.8</td>
<td>1.7</td>
<td>1.6</td>
</tr>
<tr>
<td>2) IL w/ Human Demos</td>
<td>17.6 ±0.8</td>
<td>9.7 ±0.3</td>
<td>17.5</td>
<td>9.8</td>
</tr>
<tr>
<td>3) Humans</td>
<td>87.2</td>
<td>21.8</td>
<td>89.1</td>
<td>21.9</td>
</tr>
<tr>
<td>4) IL w/ Shortest Paths</td>
<td>1.3</td>
<td>1.2</td>
<td>1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>5) IL w/ Human Demos</td>
<td>15.9 ±0.2</td>
<td>8.4 ±0.4</td>
<td>15.1</td>
<td>8.3</td>
</tr>
<tr>
<td>6) Humans</td>
<td>85.0</td>
<td>21.0</td>
<td>86.1</td>
<td>20.5</td>
</tr>
<tr>
<td>7) IL w/ Shortest Paths</td>
<td>–</td>
<td>–</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>8) IL w/ Human Demos</td>
<td>–</td>
<td>–</td>
<td>8.3</td>
<td>4.1</td>
</tr>
<tr>
<td>9) Humans</td>
<td>–</td>
<td>–</td>
<td>94.9</td>
<td>20.5</td>
</tr>
</tbody>
</table>

improvements coming from long training schedules; unfortunately, this means that wall-clock training times are not lower than RL. Fig. 2.5b shows that IL requires 7x fewer unique steps of experience to outperform success and is thus much more sample-efficient.

Zero-shot results on Gibson

[43] are in Sec. A.2.
2.6.3 PICK&PLACE

Results

We report results in Tab. 2.5 across three evaluation splits. 1) New Initializations: new locations of objects and receptacles. This tests generalization to unseen locations in seen environments. 2) New Instructions: compositionally novel object-receptacle combinations of objects and receptacles individually seen during training. 3) New Environments: generalization to 2 scenes held out from training. Similar to OBJECTNAV and as described in Sec. 2.4, we also report results with shortest paths. Again, these paths are significantly shorter (average 342 vs. 932 steps per demonstration) and hence, we generate a larger dataset of 25.7k episodes roughly matching the cumulative steps of experience with human demonstrations (8.8M shortest path steps vs. 11.5M human steps). Training on 9.5k human demonstrations achieves 17.5% success, 9.8% SPL on new object-receptacle initializations (row 2). Across splits, training on shortest paths hurts success by 8-16%. Going to new object-receptacle pairs, success drops by 2.4% (row 5 vs. 2), and then going to new environ-
ments further hurts success by 6.8% (row 8 vs. 5). We also trained an RL policy with the exploration and distance-based rewards from [16], but it failed to get beyond 0% success on new object-receptacle intializations. See the appendix (Sec. A.1.2) for training details.

Performance vs. Dataset size

Similar to OBJECTNAV, we trained policies on 2.5k to 9.5k subsets of our PICK&PLACE data, and found that performance continues to improve with more data. Figure in appendix (Sec. A.1.3).

2.7 Characterizing Learned Behaviors

To characterize the behaviors learnt by our best IL agents, we first sample 300 validation OBJECTNAV episodes for each method and manually categorize the behavior observed. A subset of observed behaviors are visualized in Fig. 2.6. Our agents demonstrate sophisticated object-search behaviors e.g. peeking into rooms to maximize sight coverage (SC), instead of occupancy coverage (OC), checking corners of rooms for small objects, beelining to goal object once seen, exhaustive search (ES), turning in place to get a panoramic view (PT), and looping back to recheck some areas. Amusingly, unlike shortest path / RL agents, these IL agents also stand idle and ‘look around’ i.e. turn in place, like humans. Tab. 2.6 quantifies these behaviors. See appendix (Sec. A.4) for details on how these were computed. Agents trained with IL on human demonstrations have higher coverage (both occupancy and sight), peaking behavior, panoramic turns, beelines, and exhaustive search than RL. RL-trained agents achieve higher average Goal Room Time Spent (GRTS) – i.e. time spent in the room

<table>
<thead>
<tr>
<th>Method</th>
<th>OC (%)</th>
<th>SC (%)</th>
<th>GRTS (%)</th>
<th>Peeks (%)</th>
<th>PT (%)</th>
<th>Beeline (%)</th>
<th>ES (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) IL w/ shortest paths</td>
<td>4.2±1.1</td>
<td>31.2±12</td>
<td>20.5±3.2</td>
<td>3.0±1.9</td>
<td>0.0±0.0</td>
<td>0.0±0.0</td>
<td>10.1±3.5</td>
</tr>
<tr>
<td>2) IL w/ human demos</td>
<td>21.4±1.8</td>
<td>72.1±15</td>
<td>22.4±1.1</td>
<td>19.6±4.4</td>
<td>4.3±2.3</td>
<td>10.3±5.1</td>
<td>55.3±5.6</td>
</tr>
<tr>
<td>3) RL [15]</td>
<td>14.6±1.6</td>
<td>66.6±5.1</td>
<td>27.7±3.5</td>
<td>9.7±5.5</td>
<td>0.0±0.0</td>
<td>0.1±2.2</td>
<td>49.0±7.0</td>
</tr>
<tr>
<td>4) Humans</td>
<td>15.4±1.6</td>
<td>70.5±3.4</td>
<td>-</td>
<td>13.8±3.9</td>
<td>5.1±2.4</td>
<td>21.6±4.8</td>
<td>52.1±5.6</td>
</tr>
</tbody>
</table>

Table 2.6. Quantifying semantic exploration behaviors for IL agents trained on shortest paths (row 1) and human demonstrations (row 2), the Red Rabbit RL agent [15] (row 3), and humans (row 4).
containing the target object – but also have significantly higher variance in GRTS across scenes compared to IL agents. See appendix (Sec. A.4) for a per-scene breakdown of GRTS as well as histograms of time spent in each room (instead of just target room) when searching for a target object.

2.8 Conclusion

This thesis develops a infrastructure to collect human demonstrations at scale and using this, trained imitation learning (IL) agents on 92k+ human demonstrations for OBJECTNAV and PICK&PLACE. On OBJECTNAV, we found that IL using 70k human demonstrations outperforms RL using 240k agent-gathered trajectories, and on PICK&PLACE, IL agents get to \( \sim 18\% \) success while RL fails to get beyond 0%. Qualitatively, we found that IL agents pick up on sophisticated object-search behavior implicitly captured in human demonstrations, much more prominently than RL agents. Overall, we believe our work makes a compelling case for investing in large-scale imitation learning of human demonstrations.

Limitations. Our approach is fundamentally limited by the limitations of imitation learning as our approach uses vanilla behavior cloning with inflection weighting. Additionally, these agents trained on human demonstrations exhibit some common failure cases. Some examples of common failure cases are – reaching close to the goal object but not within goal radius and ending episode early, trying to move straight when agent is colliding and getting stuck, looping around multiple instances of the goal object and as a result, exceeding maximum episode steps, and exploring the environment and not finding the goal object. Our approach is also limited by the amount of human demonstrations we can gather and the agent architecture being trained on this dataset. Currently, we use a vanilla CNN+RNN architecture to learn imitation learning policies but we can build better architecture which make full use of the rich semantic information these human demonstrations have.
3.1 Introduction

Since the seminal work of Winograd [54], designing embodied agents that have a rich understanding of the environment they are situated in, can interact with humans (and other agents) via language, and the environment via actions has been a long-term goal in AI [55, 57, 58, 59, 5, 60, 11, 32, 61, 56]. We focus on ObjectGoal Navigation [2, 42], wherein an agent situated in a new environment is asked to navigate to any instance of an object category (‘find a plant’, ‘find a bed’, etc.); see Fig. 3.2. OBJECTNAV is simple to explain but difficult for today’s techniques to accomplish. First, the agent needs to be able to ground the tokens in the language instruction to physical objects in the environment (e.g. what does a ‘plant’ look like?). Second, the agent needs to have rich semantic priors to guide its navigation to avoid wasteful exploration (e.g. the microwave is likely to be found in the kitchen, not the washroom). Finally, it has to keep track of where it has been in its internal memory to avoid redundant search.

Humans are adept at OBJECTNAV. In Chapter 2, we collected a large-scale dataset of 80k human demonstrations for OBJECTNAV, where human subjects on Mechanical Turk teleoperated virtual robots and searched for objects in novel houses. This first provided a human baseline on OBJECTNAV of 88.9% success rate on the Matterport3D (MP3D) dataset [33] compared to 35.4% success rate of the best performing method [62]. This dataset was then used to train agents via imitation learning (specifically, behavior cloning).

While this approach achieved state-of-art results (35.4% success rate on MP3D VAL dataset),

---

1 On VAL split, for 21 object categories, and a maximum of 500 steps.
Figure 3.1. **OBJECTNAV** success rates of agents trained using behavior cloning (BC) vs. BC-pretraining followed by reinforcement learning (RL) (in blue). RL from scratch (i.e. BC=0) fails to get off-the-ground. With more BC demonstrations, BC success increases, and it transfers to even higher RL-finetuning success. But the difference between RL-finetuning vs. BC-pretraining success (in orange) plateaus and starts to decrease beyond a certain point, indicating diminishing returns with each additional BC demonstration.

It has two clear limitations. First, behavior cloning (BC) is known to suffer from poor generalization to out-of-distribution states not seen during training, since the training emphasizes imitating actions not accomplishing their goals. Second and more importantly, it is expensive and thus not scalable. Specifically, Ramrakhya et al. [62] collected 80k demonstrations on 56 scenes in Matterport3D Dataset, which took $\sim 2894$ hours of human teleoperation and $50k$ dollars. A few months after [62] was released, a new higher-quality dataset called HM3D-Semantics v0.1 [63] became available with 120 annotated 3D scenes, and a few months after that HM3D-Semantics v0.2 added 96 additional scenes. Scaling Ramrakhya et al.’s approach to continuously incorporate new scenes involves replicating that entire effort again and again.

On the other hand, training with reinforcement learning (RL) is trivially scalable once annotated 3D scans are available. However, as demonstrated in Maksymets et al. [16], RL requires careful reward engineering, the reward function typically used for **OBJECTNAV**
Our primary technical contribution is PIRLNav, an approach for pretraining with BC and finetuning with RL for OBJECTNAV. BC pretrained policies provide a reasonable starting point for ‘bootstrapping’ RL and make the optimization easier than learning from scratch. In fact, we show that BC pretraining even unlocks RL with sparse rewards. Sparse rewards are simple (do not involve any reward engineering) and do not suffer from the unintended consequences described above. However, learning from scratch with sparse rewards is typically out of reach since most random action trajectories result in no positive rewards.

While combining IL and RL has been studied in prior work [20, 64, 65, 66, 67], the main technical challenge in the context of modern neural networks is that imitation pretraining results in weights for the policy (or actor), but not a value function (or critic). Thus, naively initializing a new RL policy with these BC-pretrained policy weights often leads to catastrophic failures due to destructive policy updates early on during RL training, especially for actor-critic RL methods [68]. To overcome this challenge, we present a two-stage learning scheme involving a critic-only learning phase first that gradually transitions over to training both the actor and critic. We also identify a set of practical recommendations for this recipe to be applied to OBJECTNAV. This leads to a PIRLNav policy that advances the state-the-art on OBJECTNAV from 60.0% success rate (in [17]) to 65.0% (+5.0%, 8.3%
Next, using this BC→RL training recipe, we conduct an empirical analysis of design choices. Specifically, an ingredient we investigate is whether human demonstrations can be replaced with ‘free’ (automatically generated) sources of demonstrations for **OBJECTNAV**, *e.g.* (1) shortest paths (SP) between the agent’s start location and the closest object instance, or (2) task-agnostic frontier exploration [69] (FE) of the environment followed by shortest path to goal-object upon observing it. We ask and answer the following:

1. ‘*Do human demonstrations capture any unique **OBJECTNAV**-specific behaviors that shortest paths and frontier exploration trajectories do not?’* Yes. We find that BC / BC→RL on human demonstrations outperforms BC / BC→RL on shortest paths and frontier exploration trajectories respectively. When we control the number of demonstrations from each source such that BC success on **TRAIN** is the same, RL-finetuning when initialized from BC on human demonstrations still outperforms the other two.

2. ‘*How does performance after RL scale with BC dataset size?’* We observe diminishing returns from RL-finetuning as we scale BC dataset size. This suggests, by effectively leveraging the trade-off curve between size of pretraining dataset size vs. performance after RL-Finetuning, we can achieve closer to state-of-the-art results without investing into a large dataset of BC demonstrations.

3. ‘*Does BC on frontier exploration demonstrations present similar scaling behavior as BC on human demonstrations?’* No. We find that as we scale frontier exploration demonstrations past 70k trajectories, the performance plateaus.

Finally, we present an analysis of the failure modes of our **OBJECTNAV** policies and present a set of guidelines for further improving them. Our policy’s primary failure modes are: a) Dataset issues: comprising of missing goal annotations, and navigation meshes blocking the path, b) Navigation errors: primarily failure to navigate between floors, c) Recognition failures: where the agent does not identify the goal object during an episode, or confuses the specified goal with a semantically-similar object.
3.2 Related Work

**ObjectGoal Navigation.** Prior works on OBJECTNAV have used end-to-end RL [70, 15, 16], modular learning [17, 18, 71], and imitation learning [62, 72]. Works that use end-to-end RL have proposed improved visual representations [70, 73], auxiliary tasks [15], and data augmentation techniques [16] to improve generalization to unseen environments. Improved visual representations include object relation graphs [73] and semantic segmentations [70]. Ye et al. [15] use auxiliary tasks like predicting environment dynamics, action distributions, and map coverage in addition to OBJECTNAV and achieve promising results. Maksymets et al. [16] improve generalization of RL agents by training with artificially inserted objects and proposing a reward to incentivize exploration.

Modular learning methods for OBJECTNAV have also emerged as a strong competitor [17, 18, 19]. These methods rely on separate modules for semantic mapping that build explicit structured map representations, a high-level semantic exploration module that is learned through RL to solve the ‘where to look?’ subproblem, and a low-level navigation policy that solves ‘how to navigate to \((x, y)\)?’.

The current state-of-the-art methods on OBJECTNAV [62, 72] make use of BC on a large dataset of \(80k\) human demonstrations with a simple CNN+RNN policy architecture. In this work, we improve on them by developing an effective approach to finetune these imitation-pretrained policies with RL.

**Imitation Learning and RL Finetuning.** Prior works have considered a special case of learning from demonstration data. These approaches initialize policies trained using behavior cloning, and then fine-tune using on-policy reinforcement learning [20, 65, 66, 67, 74, 75]. On classical tasks like cart-pole swing-up [20], balance, hitting a baseball [74], and underactuated swing-up [75], demonstrations have been used to speed up learning by initializing policies pretrained on demonstrations for RL. Similar to these methods, we also use a on-policy RL algorithm for finetuning the policy trained with behavior
cloning. Rajeswaran et al. [65] (DAPG) pretrain a policy using behavior cloning and use an augmented RL finetuning objective to stay close to the demonstrations which helps reduce sample complexity. Unfortunately DAPG is not feasible in our setting as it requires solving a systems research problem to efficiently incorporate replaying demonstrations and collecting experience online at our scale. [65] show results of the approach on a dexterous hand manipulation task with a small number of demonstrations that can be loaded in system memory and therefore did not need to solve this system challenge. This is not possible in our setting, just the $256 \times 256$ RGB observations for the $77k$ demos we collect would occupy over 2 TB memory, which is out of reach for all but the most exotic of today’s systems. There are many methods for incorporating demonstrations/imitation learning with off-policy RL [76, 77, 78, 79, 80]. Unfortunately these methods were not designed to work with recurrent policies and adapting off-policy methods to work with recurrent policies is challenging [81]. See the Sec. A.7 for more details. The RL finetuning approach that demonstrates results with an actor-critic and high-dimensional visual observations, and is thus most closely related to our setup is proposed in VPT [66]. Their approach uses Phasic Policy Gradients (PPG) [82] with a KL-divergence loss between the current policy and the frozen pretrained policy, and decays the KL loss weight $\rho$ over time to enable exploration during RL finetuning. Our approach uses Proximal Policy Gradients (PPO) [83] instead of PPG, and therefore does not require a KL constraint, which is compute-expensive, and performs better on OBJECTNAV.

3.3 OBJECTNAV and Imitation Learning

3.3.1 OBJECTNAV

In OBJECTNAV an agent is tasked with searching for an instance of the specified object category (e.g., ‘bed’) in an unseen environment. The agent must perform this task using only egocentric perceptions. Specifically, a RGB camera, Depth sensor$^2$, and a GPS+Compass

$^2$We don’t use this sensor as we don’t find it helpful.
sensor that provides location and orientation relative to the start position of the episode. The action space is discrete and consists of \textsc{move\_forward} (0.25m), \textsc{turn\_left} (30°), \textsc{turn\_right} (30°), \textsc{look\_up} (30°), \textsc{look\_down} (30°), and \textsc{stop} actions. An episode is considered successful if the agent stops within 1m Euclidean distance of the goal object within 500 steps and is able to view the object by taking turn actions [42].

We use scenes from the HM3D-Semantics v0.1 dataset [63]. The dataset consists of 120 scenes and 6 unique goal object categories. We evaluate our agent using the train/val/test splits from the 2022 Habitat Challenge\(^3\).

### 3.3.2 \textsc{ObjectNav} Demonstrations

In Chapter 2, we collected \textsc{ObjectNav} demonstrations for the Matterport3D dataset [33]. We begin our study by replicating this effort and collect demonstrations for the HM3D-Semantics v0.1 dataset [63]. We collect 77k demonstrations, amounting to ~2378 human annotation hours.

### 3.3.3 Imitation Learning from Demonstrations

We use behavior cloning to pretrain our \textsc{ObjectNav} policy on the human demonstrations we collect. Let \( \pi_{BC}^{\theta}(a_t | o_t) \) denote a policy parametrized by \( \theta \) that maps observations \( o_t \) to a distribution over actions \( a_t \). Let \( \tau \) denote a trajectory consisting of state, observation, action tuples: \( \tau = (s_0, o_0, a_0, \ldots, s_T, o_T, a_T) \) and \( \mathcal{T} = \{ \tau^{(i)} \}_{i=1}^{N} \) denote a dataset of human demonstrations. The optimal parameters are

\[
\theta^* = \arg \min_{\theta} \sum_{i=1}^{N} \sum_{(o_t, a_t) \in \tau^{(i)}} - \log \left( \pi_{BC}^{\theta}(a_t | o_t) \right) \tag{3.1}
\]

We use inflection weighting [9] to adjust the loss function to upweight timesteps where actions change (i.e. \( a_{t-1} \neq a_t \)).

\(^3\)https://aihabitat.org/challenge/2022/
Our **ObjectNav policy** architecture is a simple CNN+RNN model from [72]. To encode RGB input \((i_t = \text{CNN}(I_t))\), we use a ResNet50 [44]. Following [72], the CNN is first pre-trained on the Omnidata starter dataset [84] using the self-supervised pretraining method DINO [85] and then finetuned during ObjectNav training. The GPS+Compass inputs, \(P_t = (\Delta x, \Delta y, \Delta z)\), and \(R_t = (\Delta \theta)\), are passed through fully-connected layers \(p_t = \text{FC}(P_t), r_t = \text{FC}(R_t)\) to embed them to 32-d vectors. Finally, we convert the object goal category to one-hot and pass it through a fully-connected layer \(g_t = \text{FC}(G_t)\), resulting in a 32-d vector. All of these input features are concatenated to form an observation embedding, and fed into a 2-layer, 2048-d GRU at every timestep to predict a distribution over actions \(a_t\) - formally, given current observations \(o_t = [i_t, p_t, r_t, g_t]\), \((h_t, a_t) = \text{GRU}(o_t, h_{t-1})\). To reduce overfitting, we apply color-jitter and random shifts [86] to the RGB inputs.

### 3.4 RL Finetuning

Our motivation for RL-finetuning is two-fold. First, finetuning may allow for higher performance as behavior cloning is known to suffer from a train/test mismatch – when training, the policy sees the result of taking ground-truth actions, while at test-time, it must contend with the consequences of its own actions. Second, collecting more human demonstrations on new scenes or simply to improve performance is time-consuming and expensive. On the other hand, RL-finetuning is trivially scalable (once annotated 3D scans are available) and has the potential to reduce the amount of human demonstrations needed.

#### 3.4.1 Setup

The RL objective is to find a policy \(\pi_\theta(a|s)\) that maximizes expected sum of discounted future rewards. Let \(\tau\) be a sequence of object, action, reward tuples \((o_t, a_t, r_t)\) where \(a_t \sim \pi_\theta(\cdot | o_t)\) is the action sampled from the agent’s policy, and \(r_t\) is the reward. For a
discount factor $\gamma$, the optimal policy is

$$
\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi}[R_T], \text{ where } R_T = \sum_{t=1}^{T} \gamma^{t-1} r_t.
$$

(3.2)

To solve this maximization problem, actor-critic RL methods learn a state-value function $V(s)$ (also called a critic) in addition to the policy (also called an actor). The critic $V(s_t)$ represents the expected value of returns $R_t$ when starting from state $s_t$ and acting under the policy $\pi$, where returns are defined as $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. We use DD-PPO [13], a distributed implementation of PPO [83], an on-policy RL algorithm. Given a $\theta$-parameterized policy $\pi_{\theta}$ and a set of rollouts, PPO updates the policy as follows. Let $\hat{A}_t = R_t - V(s_t)$, be the advantage estimate and $p_t(\theta) = \frac{\pi_{\theta}(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}$ be the ratio of the probability of action $a_t$ under current policy and under the policy used to collect rollouts. The parameters are updated by maximizing:

$$
J_{PPO}(\theta) = \mathbb{E}_t \left[ \min(p_t(\theta)\hat{A}_t, \text{clip}(p_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right]
$$

(3.3)

We use a sparse success reward. Sparse success is simple (does not require hyperparameter optimization) and has fewer unintended consequences (e.g. Maksymets et al. [16] showed that typical dense rewards used in OBJECTNAV actually penalize exploration, even though exploration is necessary for OBJECTNAV in new environments). Sparse rewards are desirable but typically difficult to use with RL (when initializing training from scratch) because they result in nearly all trajectories achieving 0 reward, making it difficult to learn. However, since we pretrain with BC, we do not observe any such pathologies.

3.4.2 Finetuning Methodology

We use the behavior cloned policy $\pi_{\theta}^{BC}$ weights to initialize the actor parameters. However, notice that during behavior cloning we do not learn a critic nor is it easy to do so – a critic learned on human demonstrations (during behavior cloning) would be overly optimistic
since all it sees are successes. Thus, we must learn the critic from scratch during RL. Naively finetuning the actor with a randomly-initialized critic leads to a rapid drop in performance\(^4\) (see Fig. A.8) since the critic provides poor value estimates which influence the actor’s gradient updates (see Eq.(Equation 3.3)). We address this issue by using a two-phase training regime:

**Phase 1: Critic Learning.** In the first phase, we rollout trajectories using the frozen policy, pre-trained using BC, and use them to learn a critic. To ensure consistency of rollouts collected for critic learning with RL training, we sample actions (as opposed to using \(\text{argmax}\) actions) from the pre-trained BC policy: \(a_t \sim \pi_\theta(s_t)\). We train the critic until its loss plateaus. In our experiments, we found \(8M\) steps to be sufficient. In addition, we also initialize the weights of the critic’s final linear layer close to zero to stabilize training.

**Phase 2: Interactive Learning.** In the second phase, we unfreeze the actor RNN\(^5\) and finetune both actor and critic weights. We find that naively switching from phase 1 to phase

---

\(^4\)After the initial drop, the performance increases but the improvements on success are small.

\(^5\)The CNN and non-visual observation embedding layers remain frozen. We find this to be more stable.
2 leads to small improvements in policy performance at convergence. We gradually decay the critic learning rate from $2.5 \times 10^{-4}$ to $1.5 \times 10^{-5}$ while warming-up the policy learning rate from 0 to $1.5 \times 10^{-5}$ between $8M$ to $12M$ steps, and then keeping both at $1.5 \times 10^{-5}$ through the course of training. See Fig. 3.3. We find that using this learning rate schedule helps improve policy performance. For parameters that are shared between the actor and critic (i.e. the RNN), we use the lower of the two learning rates (i.e. always the actor’s in our schedule). To summarize our finetuning methodology:

- First, we initialize the weights of the policy network with the IL-pretrained policy and initialize critic weights close to zero. We freeze the actor and shared weights. The only learnable parameters are in the critic.
- Next, we learn the critic weights on rollouts collected from the pretrained, frozen policy.
- After training the critic, we warmup the policy learning rate and decay the critic learning rate.
- Once both critic and policy learning rate reach a fixed learning rate, we train the policy to convergence.

3.4.3 Results

**Comparing with the RL-finetuning approach in VPT [66].** We start by comparing our proposed RL-finetuning approach with the approach used in VPT [66]. Specifically, [66] proposed initializing the critic weights to zero, replacing entropy term with a KL-divergence loss between the frozen IL policy and the RL policy, and decay the KL divergence loss coefficient, $\rho$, by a fixed factor after every iteration. Notice that this prevents the actor from drifting too far too quickly from the IL policy, but does not solve the uninitialized critic problem. To ensure fair comparison, we implement this method within our DD-PPO framework to ensure that any performance difference is due to the fine-tuning algorithm and not tangential implementation differences. Complete training details are in the Sec. A.9.3. We keep hyperparameters constant for our approach for all experiments. Tab. 3.1 reports
results on HM3D VAL for the two approaches using 20$k human demonstrations. We find that PIRLNav achieves +2.2% Success compared to VPT and comparable SPL.

### Table 3.1. Comparison with VPT on HM3D VAL [19, 63]

<table>
<thead>
<tr>
<th>Method</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) BC</td>
<td>52.0</td>
<td>20.6</td>
</tr>
<tr>
<td>2) BC→RL-FT w/ VPT</td>
<td>59.7 ±0.70</td>
<td>28.6 ±0.89</td>
</tr>
<tr>
<td>3) PIRLNav (Ours)</td>
<td>61.9 ±0.47</td>
<td>27.9 ±0.56</td>
</tr>
</tbody>
</table>

### Table 3.2. RL-finetuning ablations on HM3D VAL [19, 63]

<table>
<thead>
<tr>
<th>Method</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) BC</td>
<td>52.0</td>
<td>20.6</td>
</tr>
<tr>
<td>2) BC→RL-FT</td>
<td>53.6 ±1.01</td>
<td>28.6 ±0.50</td>
</tr>
<tr>
<td>3) BC→RL-FT (+ Critic Learning)</td>
<td>56.7 ±0.93</td>
<td>27.7 ±0.82</td>
</tr>
<tr>
<td>4) BC→RL-FT (+ Critic Learning, Critic Decay)</td>
<td>59.4 ±0.42</td>
<td>26.9 ±0.38</td>
</tr>
<tr>
<td>5) BC→RL-FT (+ Critic Learning, Actor Warmup)</td>
<td>58.2 ±0.55</td>
<td>26.7 ±0.60</td>
</tr>
<tr>
<td>6) PIRLNav (Ours)</td>
<td>61.9 ±0.47</td>
<td>27.9 ±0.56</td>
</tr>
</tbody>
</table>

**Ablations.** Next, we conduct ablation experiments to quantify the importance of each phase in our RL-finetuning approach. Tab. 3.2 reports results on the HM3D VAL split for a policy BC-pretrained on 20$k human demonstrations and RL-finetuned for 300$M$ steps, complete training details are in Sec. A.9.4. First, without a gradual learning transition (row 2), i.e. without a critic learning and LR decay phase, the policy improves by 1.6% on success and 8.0% on SPL. Next, with only a critic learning phase (row 3), the policy improves by 4.7% on success and 7.1% on SPL. Using an LR decay schedule only for the critic after the critic learning phase improves success by 7.4% and SPL by 6.3%, and using an LR warmup schedule for the actor (but no critic LR decay) after the critic learning phase improves success by 6.2% and SPL by 6.1%. Finally, combining everything (critic-only learning, critic LR decay, actor LR warmup), our policy improves by 9.9% on success and 7.3% on SPL.

**ObjectNav Challenge 2022 Results.** Using our overall two-stage training approach of

---

*The approach is called “BadSeed” on the HM3D leaderboard: eval.ai/web/challenges/challenge-page/1615/leaderboard/3899*
Table 3.3. Results on HM3D TEST-STANDARD and TEST-CHALLENGE [87, 63]. Unpublished works submitted only to the OBJECTNAV leaderboard have been grayed out.

<table>
<thead>
<tr>
<th>Method</th>
<th>TEST-STD</th>
<th></th>
<th>TEST-CHALLENGE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Success (↑)</td>
<td>SPL (↑)</td>
<td>Success (↑)</td>
<td>SPL (↑)</td>
</tr>
<tr>
<td>1) Stretch [17]</td>
<td>60.0%</td>
<td>34.0%</td>
<td>56.0%</td>
<td>29.0%</td>
</tr>
<tr>
<td>2) ProcTHOR-Large [88]</td>
<td>54.0%</td>
<td>32.0%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3) Habitat-Web [62]</td>
<td>55.0%</td>
<td>22.0%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4) DD-PPO [87]</td>
<td>26.0%</td>
<td>12.0%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5) Populus A.</td>
<td>66.0%</td>
<td>32.0%</td>
<td>60.0%</td>
<td>30.0%</td>
</tr>
<tr>
<td>6) ByteBOT</td>
<td>68.0%</td>
<td>37.0%</td>
<td>64.0%</td>
<td>35.0%</td>
</tr>
<tr>
<td>7) PIRLNav⁶</td>
<td>65.0%</td>
<td>33.0%</td>
<td>65.0%</td>
<td>33.0%</td>
</tr>
</tbody>
</table>

BC-pretraining followed by RL-finetuning, we achieve state-of-the-art results on OBJECTNAV—65.0% success and 33.0% SPL on both the TEST-STANDARD and TEST-CHALLENGE splits and 70.4% success and 34.1% SPL on VAL. Tab. 3.3 compares our results with the top-4 entries to the Habitat OBJECTNAV Challenge 2022 [87]. Our approach outperforms Stretch [17] on success rate on both TEST-STANDARD and TEST-CHALLENGE and is comparable on SPL (1% worse on TEST-STANDARD, 4% better on TEST-CHALLENGE). ProcTHOR [88], which uses 10k procedurally-generated environments for training, achieves 54% success and 32% SPL on TEST-STANDARD split, which is 11% worse at success and 1% worse at SPL than ours. For sake of completeness, we also report results of two unpublished entries uploaded to the leaderboard—Populus A. and ByteBOT. Unfortunately, there is no associated report yet with these entries, so we are unable to comment on the details of these approaches, or even whether the comparison is meaningful.

3.5 Role of demonstrations in BC→RL transfer

Our decision to use human demonstrations for BC-pretraining before RL-finetuning was motivated by results in prior work [62]. Next, we examine if other cheaper sources of demonstrations lead to equally good BC→RL generalization. Specifically, we consider 3
sources of demonstrations:

**Shortest paths (SP).** These demonstrations are generated by greedily sampling actions to fit the geodesic shortest path to the nearest navigable goal object, computed using the ground-truth map of the environment. These demonstrations do not capture any exploration, they only capture success at the **OBJECTNAV** task via the most efficient path.

**Task-Agnostic Frontier Exploration (FE) [17].** These are generated by using a 2-stage approach: 1) Exploration: where a task-agnostic strategy is used to maximize exploration coverage and build a top-down semantic map of the environment, and 2) Goal navigation: once the goal object is detected by the semantic predictor, the developed map is used to reach it by following the shortest path. These demonstrations capture **OBJECTNAV**-agnostic exploration.

**Human Demonstrations (HD) [62].** These are collected by asking humans on Mechanical Turk to control an agent and navigate to the goal object. Humans are provided access to the first-person RGB view of the agent and tasked to reach within 1m of the goal object category. These demonstrations capture human-like **OBJECTNAV**-specific exploration.

### 3.5.1 Results with Behavior Cloning

Using the BC setup described in Sec. 3.3.3, we train on SP, FE, and HD demonstrations. Since these demonstrations vary in trajectory length (*e.g.* SP are significantly shorter than FE), we collect $\sim12M$ steps of experience with each method. That amounts to $240k$ SP, $70k$ FE, and $77k$ HD demonstrations respectively. As shown in Tab. 3.4, BC on $240k$ SP demonstrations leads to 6.4% success and 5.0% SPL. We believe this poor performance is due to an imitation gap [89], *i.e.* the shortest path demonstrations are generated with access to privileged information (ground-truth map of the environment) which is not available to the policy during training. Without a map, following the shortest path in a new environment to find a goal object is not possible. BC on $70k$ FE demonstrations achieves 44.9% success and 21.5% SPL, which is significantly better than BC on shortest paths (+38.5% success,
Table 3.4. Performance on HM3D VAL with imitation learning on SP, FE, and HD demonstrations. The size of each demonstration dataset is picked such that total steps of experience is $\sim 12M$.

<table>
<thead>
<tr>
<th>Training demonstrations</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest paths (240k)</td>
<td>6.4%</td>
<td>5.0%</td>
</tr>
<tr>
<td>Frontier exploration (70k)</td>
<td>44.9%</td>
<td>21.5%</td>
</tr>
<tr>
<td>Human demonstrations (77k)</td>
<td>64.1%</td>
<td>27.1%</td>
</tr>
</tbody>
</table>

Finally, BC on 77k HD obtains the best results – 64.1% success, 27.1% SPL. These trends suggest that task-specific exploration (captured in human demonstrations) leads to much better generalization than task-agnostic exploration (FE) or shortest paths (SP).

3.5.2 Results with RL Finetuning

![Figure 3.4](image)

**Figure 3.4.** OBJECTNAV performance on HM3D VAL with BC-pretraining on shortest path (SP), frontier exploration (FE), and human demonstrations (HD), followed by RL-finetuning from each.

Using the BC-pretrained policies on SP, FE, and HD demonstrations as initialization, we RL-finetune each using our approach described in Sec. 3.4. These results are summarized in Fig. 3.4. Perhaps intuitively, the trends after RL-finetuning follow the same ordering as BC-pretraining, i.e., RL-finetuning from BC on HD $>$ FE $>$ SP. But there are two factors that could be leading to this ordering after RL-finetuning – 1) inconsistency in performance at initialization (i.e. BC on HD is already better than BC on FE), and 2) amenability of each of these initializations to RL-finetuning (i.e. is RL-finetuning from HD init better than FE init?).
We are interested in answering (2), and so we control for (1) by selecting BC-pretrained policy weights across SP, FE, and HD that have equal performance on a subset of TRAIN = \( \sim 48.0\% \) success. This essentially amounts to selecting BC-pretraining checkpoints for FE and HD from earlier in training as \( \sim 48.0\% \) success is the maximum for SP.

Fig. 3.5 shows the results after BC and RL-finetuning on a subset of the HM3D TRAIN and on HM3D VAL. First, note that at BC-pretraining TRAIN success rates are equal (\( = \sim 48.0\% \)), while on VAL FE is slightly better than HD followed by SP. We find that after RL-finetuning, the policy trained on HD still leads to higher VAL success (66.1\%) compared to FE (51.3\%) and SP (43.6\%). Notice that RL-finetuning from SP leads to high TRAIN success, but low VAL success, indicating significant overfitting. FE has smaller TRAIN-VAL gap after RL-finetuning but both are worse than HD, indicating underfitting. These results show that learning to imitate human demonstrations equips the agent with navigation strategies that enable better RL-finetuning generalization compared to imitating other kinds of demonstrations, even when controlled for the same BC-pretraining accuracy.

**Results on SP-favoring and FE-favoring episodes.** To further emphasize that imitating human demonstrations is key to good generalization, we created two subsplits from the
Table 3.5. Results on SP-favoring and FE-Favoring splits.

<table>
<thead>
<tr>
<th>Training demonstrations</th>
<th>BC Success (↑)</th>
<th>RL-FT Success (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) SP</td>
<td>5.2%</td>
<td>34.8%</td>
</tr>
<tr>
<td>2) HD</td>
<td>0.0%</td>
<td>57.2%</td>
</tr>
<tr>
<td>3) FE</td>
<td>26.3%</td>
<td>43.0%</td>
</tr>
<tr>
<td>4) HD</td>
<td>0.0%</td>
<td>57.2%</td>
</tr>
</tbody>
</table>

HM3D val split that are adversarial to HD performance – SP-favoring and FE-favoring. The SP-favoring val split consists of episodes where BC on SP achieved a higher performance compared to BC on HD, i.e. we select episodes where BC on SP succeeded but BC on HD did not or both BC on SP and BC on HD failed. Similarly, we also create an FE-favoring val split using the same sampling strategy biased towards BC on FE. Next, we report the performance of RL-finetuned from BC on SP, FE, and HD on these two evaluation splits in Tab. 3.5. On both SP-favoring and FE-favoring, BC on HD is at 0% success (by design), but after RL-finetuning, is able to significantly outperform RL-finetuning from the respective BC on SP and FE policies.

3.5.3 Scaling laws of BC and RL

In this section, we investigate how BC-pretraining $\rightarrow$ RL-finetuning success scales with no. of BC demonstrations.

**Human demonstrations.** We create HD subsplits ranging in size from $2k$ to $77k$ episodes, and BC-pretrain policies with the same set of hyperparameters on each split. Then, for each, we RL-finetune from the best-performing checkpoint. The resulting BC and RL success on HM3D val vs. no. of HD episodes is plotted in Fig. 3.1. Similar to [62], we see promising scaling behavior with more BC demonstrations.

Interestingly, as we increase the size of of the BC pretraining dataset and get to high BC accuracies, the improvements from RL-finetuning decrease. E.g. at $20k$ BC demonstrations, the BC$\rightarrow$RL improvement is 10.1% success, while at $77k$ BC demonstrations, the improvement
Figure 3.6. Success on ObjectNav HM3D VAL split vs. no. of frontier exploration demonstrations for training.

is 6.3%. Furthermore, with 35k BC-pretraining demonstrations, the RL-finetuned success is only 4% worse than RL-finetuning from 77k BC demonstrations (66.4% vs. 70.4%). Both suggest that by effectively leveraging the trade-off between the size of the BC-pretraining dataset vs. performance gains after RL-finetuning, it may be possible to achieve close to state-of-the-art results without large investments in demonstrations.

**How well does FE Scale?** In Sec. 3.5.1, we showed that BC on human demonstrations outperforms BC on both shortest paths and frontier exploration demonstrations, when controlled for the same amount of training experience. In contrast to human demonstrations however, collecting shortest paths and frontier exploration demonstrations is cheaper, which makes scaling these demonstration datasets easier. Since BC performance on shortest paths is significantly worse even with 3x more demonstrations compared to FE and HD (240k SP vs. 70k FE and 77k HD demos, Sec. 3.5.1), we focus on scaling FE demonstrations. Fig. 3.6 plots performance on HM3D VAL against FE dataset size and a curve fitted using 75k demonstrations to predict performance on FE dataset-sizes ≥ 75k. We created splits
ranging in size from $10k$ to $150k$. Increasing the dataset size doesn’t consistently improve performance and saturates after $70k$ demonstrations, suggesting that generating more FE demonstrations is unlikely to help. We hypothesize that the saturation is because these demonstrations don’t capture task-specific exploration.

### 3.6 Failure Modes

To better understand the failure modes of our BC→RL OBJECTNAV policies, we manually annotate 592 failed HM3D VAL episodes from our best OBJECTNAV agent. See Fig. 3.7. The most common failure modes are:

- **Missing Annotations** (27%): Episodes where the agent navigates to the correct goal object category but the episode is counted as a failure due to missing annotations in the data.
- **Inter-Floor Navigation** (21%): The object is on a different floor and the agent fails to climb up/down the stairs.
- **Recognition Failure** (20%): The agent sees the object in its field of view but fails to navigate to it.
- **Last Mile Navigation [90]** (12%). Repeated collisions against objects or mesh geometry close to the goal object preventing the agent from reaching close to it.
- **Navmesh Failure** (9%). Hard-to-navigate meshes blocking the path of the agent. E.g. in one instance, the agent fails to climb stairs because of a narrow nav mesh on the stairs.
- **Looping** (4%). Repeatedly visiting the same location and not exploring the rest of the
environment.

**Semantic Confusion** (5%). Confusing the goal object with a semantically-similar object. *E.g.* ‘armchair’ for ‘sofa’.

**Exploration Failure** (2%). Catch-all for failures in a complex navigation environment, early termination, semantic failures (*e.g.* looking for a chair in a bathroom), *etc.*

As can be seen in Fig. 3.7, most failures (~36%) are due to issues in the **OBJECTNAV** dataset – 27% due to missing object annotations + 9% due to holes / issues in the navmesh. 21% failures are due to the agent being unable to climb up/down stairs. We believe this happens because climbing up / down stairs to explore another floor is a difficult behavior to learn and there are few episodes that require this. Oversampling inter-floor navigation episodes during training can help with this. Another failure mode is failing to recognize the goal object – 20% where the object is in the agent’s field of view but it does not navigate to it, and 5% where the agent navigates to another semantically-similar object. Advances in the visual backbone and object recognition can help address these. Prior works [62, 17] have used explicit semantic segmentation modules to recognize objects at each step of navigation. Incorporating this within the BC→RL training pipeline could help. 11% failures are due to last mile navigation, suggesting that equipping the agent with better goal-distance estimators could help. Finally, only ~6% failures are due to looping and lack of exploration, which is promising!

### 3.7 Conclusion

To conclude, we propose PIRLNav, an approach to combine imitation using behavior cloning (BC) and reinforcement learning (RL) for **OBJECTNAV**, wherein we pretrain a policy with BC on 77$k$ human demonstrations and then finetune it with RL, leading to state-of-the-art results on **OBJECTNAV** (65% success, 5% improvement over previous best). Next, using this BC→RL training recipe, we present a thorough empirical study of the impact of different demonstration datasets used for BC-pretraining on downstream RL-finetuning performance.
We show that BC / BC→RL on human demonstrations outperforms BC / BC→RL on shortest paths and frontier exploration trajectories, even when we control for same BC success on TRAIN. We also show that as we scale the pretraining dataset size for BC and get to higher BC success rates, the improvements from RL-finetuning start to diminish. Finally, we characterize our agent’s failure modes, and find that the largest sources of error are 1) dataset annotation noise, and inability of the agent to 2) navigate across floors, and 3) recognize the correct goal object.
In this thesis, we advance the agenda of building intelligent agents that can navigate to objects by leveraging simulation and embodied demonstrations from humans. In Habitat-Web (Part I), we developed the infrastructure to collect human demonstrations at scale and using this, trained imitation learning (IL) agents on $92k+$ human demonstrations for OBJECTNAV and PICK&PLACE. On OBJECTNAV, we found that IL using $70k$ human demonstrations outperforms RL using $240k$ agent-gathered trajectories, and on PICK&PLACE, IL agents get to $\sim 18\%$ success while RL fails to get beyond $0\%$. Qualitatively, we found that IL agents pick up on sophisticated object-search behavior implicitly captured in human demonstrations, much more prominently than RL agents. Next, we propose PIRLNav (Part II), an approach to combine imitation using behavior cloning (BC) and reinforcement learning (RL) for OBJECTNAV, wherein we pretrain a policy with BC on $77k$ human demonstrations and then finetune it with RL, leading to state-of-the-art results on OBJECTNAV ($65\%$ success, $5\%$ improvement over previous best). Using this BC→RL training recipe, we present a thorough empirical study of the impact of different demonstration datasets used for BC-pretraining on downstream RL-finetuning performance. We show that BC / BC→RL on human demonstrations outperforms BC / BC→RL on shortest paths and frontier exploration trajectories, even when we control for same BC success on TRAIN. We also show that as we scale the pretraining dataset size for BC and get to higher BC success rates, the improvements from RL-finetuning start to diminish. Finally, we characterize our agent’s failure modes, and present guidelines for further improving them.
REFERENCES


[89] L. Weihs et al., “Bridging the imitation gap by adaptive insubordination”, in NeurIPS, the first two authors contributed equally, 2021.


Appendices
APPENDIX A
ADDITIONAL DETAILS AND RESULTS

A.1 Pick&Place

Recall that in the pick-and-place task (PICK&PLACE), an agent must follow an instruction of the form ‘Place the <object> on the <receptacle>’, without being told the location of the <object> or <receptacle> in a new environment. The agent must explore and navigate to the object, pick it up, explore and navigate to the receptacle, and place the previously picked-up object on it. In this section, we go over statistics of the human demonstrations dataset, how our PICK&PLACE imitation learning (IL) agents scale as a function of training dataset size, and details of our reinforcement learning baseline for PICK&PLACE.

A.1.1 Dataset Stats

Fig. A.2 compares the episode length and action histograms for human and shortest path demonstrations for PICK&PLACE. Human demonstrations are longer (average 932 vs 342 steps per demonstration) and have a more uniform action distribution compared to shortest paths. Human demonstrations also make use of all 9 actions whereas shortest path demonstrations use only 6 actions. Notice, humans also tend to stand idle and do nothing (50ms of idle time is translated to a NO_OP action). They likely use this time to strategize their next set of actions to explore the environment, which is not the case in shortest path demonstrations (by design).

A.1.2 RL Baseline

Similar to the imitation learning baseline, our base policy is a simple CNN+RNN architecture. We first embed all sensory inputs using feed-forward modules. For RGB, we use a randomly initialized ResNet18 [44]. For depth, we use a ResNet50 that was pretrained on PointGoal
navigation using DDPPO [13]. In addition to RGBD observations, the policy gets as input language instructions of the form ‘Place the <object> on the <receptacle>’ encoded using a single-layer LSTM [49]. RGBD and instruction features are concatenated to form an observation embedding, which is fed into a 2-layer, 512-d GRU at every timestep. We train this policy for ∼100M steps on ∼9.5k episodes.

Rewards. The agent receives a sparse success reward $r_{\text{success}}$, a slack reward $r_{\text{slack}}$ to motivate faster goal-seeking, an exploration reward $r_{\text{explore}}$, an object seen reward $r_{\text{seen}}$, a grab/release success reward $r_{\text{grab\_release}}$, and a drop penalty reward $r_{\text{drop\_penalty}}$ to penalize dropping the object far from the receptacle. For incentivizing exploration, we use a visitation-based coverage reward from Ye et al. [15]. We first divide the map into a voxel grid of $2.5m \times 2.5m \times 2.5m$ voxels and reward the agent for visiting each voxel. Similar to [15], we smooth $r_{\text{explore}}$ by decaying it by number of steps the agent has spent in the voxel (visit count $v$). To ensure that the agent prioritizes PICK&PLACE (and not just exploration), we decay $r_{\text{explore}}$ based on episode timestep $t$ with a decay constant of $d = 0.995$. The agent is provided a reward for exploration until it sees the object. Once it sees the object, it receives a significant positive reward $r_{\text{seen}}$, and then the reward switches to a path-efficiency based navigation reward. In addition, the agent also receives a significant positive reward when it
successfully grabs an object or releases the object close to the receptacle.

\[
\begin{align*}
    r_{\text{total}} &= r_{\text{success}} + r_{\text{slack}} + r_{\text{explore}} + r_{\text{grab release}} \\
                       &\quad + r_{\text{drop penalty}} + r_{\text{seen}} \\
    r_{\text{success}} &= 5.5 \quad \text{on success} \\
    r_{\text{slack}} &= -10^{-4} \quad \text{per step} \\
    r_{\text{seen}} &= 1.5 \quad \text{First time object seen} \\
    r_{\text{drop penalty}} &= -3.5 \quad \text{Object dropped > 2m away from receptacle} \\
    r_{\text{grab release}} &= 2.0 \quad \text{Grab / release success} \\
    r_{\text{explore}} &= 0.25 \times \frac{d}{v} \quad \text{Until object seen}
\end{align*}
\]  

(A.1a) \quad (A.1b) \quad (A.1c) \quad (A.1d) \quad (A.1e) \quad (A.1f) \quad (A.1g) \quad (A.1h)

**Results.** A policy trained with this reward for 100M steps fails to get beyond 0% success on the PICK\&PLACE task. The agent learns to pick up the object at the start of training if it sees the object while navigating but it fails to search for the receptacle and place the object on top of receptacle. Overall, throughout training, the agent doesn’t solve the task successfully even once demonstrating the difficulty of the task and inadequacy of the above reward structure.

**A.1.3 Performance vs. Dataset Size**

Fig. A.1 plots VAL success of our IL agent vs. the size of the PICK\&PLACE human demonstrations dataset. We trained policies on 2.5k to 9.5k subsets of the data. Performance continues to improve with more data and has not saturated.
Figure A.2. Comparison of episode lengths and action histograms for human demonstrations vs. shortest paths for Pick&Place. Human demonstrations are longer and have a more uniform action distribution than shortest paths.

A.2 Zero-shot OBJECTNAV results on Gibson

To test generalization of the IL agents trained on human demonstrations, we report zero-shot results by transferring our policy trained on 40k human demonstrations to the Gibson dataset [43] Val split in Table A.1. To enable zero-shot transfer of semantic features, we remap the common goal categories (chair, couch, potted plant, bed, toilet, TV, dining-table) from Matterport3D [33] to Gibson goal category IDs. Our IL agent achieves 49.4% success and 16.4% SPL (row 7) with no finetuning on Gibson dataset. Comparing our zero-shot results to approaches trained on Gibson, our IL agent is 33.6% better on success and 11.5% better on SPL than an RL baseline that takes RGBD + Semantics as input (row 3 vs. row 7). Next, we compare our approach with SemExp [53] which builds explicit semantic maps and learns a goal-oriented semantic exploration policy which learns semantic priors for efficient navigation. Our approach is 4.9% worse on success and 3.5% worse on SPL compared to SemExp [53] (row 6 vs. row 7). [71] uses a modular framework for OBJECTNAV by using a potential function conditioned on a semantic map which is used to decide where to look for unseen object in an environment. Our approach is 24.1% worse on success and 24.6% worse on SPL.

A.3 Estimating time using a LoCoBot motion model

To estimate the time a robot would take to execute the collected human trajectories in the real world, we use the LoCoBot motion model from Krantz et al. [30]. This model consists
Table A.1. ObjectNav results on the Gibson VAL split.

<table>
<thead>
<tr>
<th>Method</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Random</td>
<td>0.4%</td>
<td>0.4%</td>
</tr>
<tr>
<td>2) RGBD+RL [21]</td>
<td>8.2%</td>
<td>2.7%</td>
</tr>
<tr>
<td>3) RGBD+Semantics+RL [70]</td>
<td>15.9%</td>
<td>4.9%</td>
</tr>
<tr>
<td>4) Classical Map + FBE</td>
<td>40.3%</td>
<td>12.4%</td>
</tr>
<tr>
<td>5) Active Neural SLAM [91]</td>
<td>44.6%</td>
<td>14.5%</td>
</tr>
<tr>
<td>6) SemExp [53]</td>
<td>54.4%</td>
<td>19.9%</td>
</tr>
<tr>
<td>7) PONI [71]</td>
<td>73.6%</td>
<td>41.0%</td>
</tr>
<tr>
<td>8) IL w/ 70k Human Demos (Zero-Shot)</td>
<td>49.5%</td>
<td>16.4%</td>
</tr>
</tbody>
</table>

of a rotation function that maps turn angle to time and a translation function that maps straight-line distance to time. For estimating time required for grab/release actions, we replace them with 0.15m forward steps and use the straight-line distance translation function. We use the MOVEBASE controller from [30] for all our time estimates, with the following rotation and translation equations:

\[ y_{\text{rotate}} = 0.000358\phi^2 + 0.108\phi + 2.23 \]  

(A.2)

\[ y_{\text{translate}} = 4.2x + 0.362 \]  

(A.3)

A.4 Characterizing Learnt Behaviors

In this section, we describe the metrics used to characterize the exploration behavior exhibited by these agents in Sec. 7 in the main paper. These include 1) Occupancy Coverage (OC) and 2) Sight Coverage (SC) introduced in Sec. 4 in the main paper, as well as 3) Goal Room Time Spent (GRTS) – the number of steps as a fraction of total episode length an agent takes within the room bounding box containing the target object, 4) Peeks – check if the agent steps back into the last visited room after taking just ~10 steps in another room, 5) Panoramic Turn (PT) – whether the agent stands at one place and turns left and right to
get sweeping views, 6) Beeline – if the agent takes 10 continuous forward actions before reaching the goal in the last 15 steps, 7) Exhaustive Search (ES) – ≥ 75% sight coverage. To compute these metrics, we use the semantic annotations in Matterport3D. These annotations provide 3D bounding box coordinates for each room category in an environment. We use these bounding box coordinates to track the rooms an agent visits during an episode. GRTS gives us a measure of how often the agent ends up reaching goal object room but doesn’t successfully locate the object. A higher GTRS suggests that the agent is at least good at reaching semantically meaningful locations in search of the goal object. We find that RL agents have higher average GRTS but also significantly higher variance in GRTS across scenes while our IL agents have lower average GRTS but more consistently spend time in the target room (see Fig. A.3). To evaluate not just the final room the agent ends up at, but all the rooms it visits through the course of an episode, we also plot distributions of the time spent per room category for each goal object for human demonstrations vs. IL agents trained on human demonstrations vs. RL agents.
A.5 Inter-human Variance in OBJECTNav

To get a sense for the variance in OBJECTNav human demonstrations, we collected 20 unique human-provided trajectories for the same initial location and target object (‘cabinet’). This is visualized in Fig. A.4. We see that there is quite a bit of diversity in navigation trajectories across humans. They often navigate to different instances of the goal object category ‘cabinet’, and even when multiple humans go to the same object instance, the routes taken are different (red vs. blue trajectory).

We also plot the average SPL per AMT user in our dataset in Fig. A.5. We find that human performance has a lot of variability, ranging from 25.2% to 68.2% (Fig. A.5a). The SPL range that has the most humans is ~50%. The best-performing human annotator achieves an SPL of 68.2% averaged over 6 episodes (Fig. A.5b), which is particularly close to shortest paths and arguably super-human.

A.6 AMT Interface

Fig. A.6 shows a screenshot of our AMT interface for collecting PICK&PLACE demonstrations. For the PICK&PLACE task, we provide humans with an instruction of the form ‘Place the <object> on the <receptacle>’, without being told the location of the <object>
or *<receptacle>* in a new environment, and they can see agent’s first-person view of the environment. They can make the agent move, look around, and interact with the environment using keyboard controls. Once the AMT user completes the task they can submit the task by clicking the ‘Submit’ button. We then run task-specific validation checks to ensure only successful tasks get submitted.

**Validation.** To ensure data quality, every submitted AMT task goes through a set of validation checks. For *OBJECTNAV*, we use the same set of validation checks as the Habitat challenge evaluation setup, *i.e.* a task is considered successful only when the user has moved the agent to within 1 m of the goal object. We do not limit the maximum number of steps to allow users on AMT to explore the environment. This captures key human exploration behavior necessary to succeed at these tasks. Similarly, for *PICK&PLACE*, a task is considered successful when the target object is placed on a receptacle object. Specifically, we check if the Euclidean distance between the centers of the target and receptable objects is less than 0.7 m, and that the target object is at a height greater than the receptacle center.

---

**Figure A.5.** a) Histogram of average SPL for each AMT user for *OBJECTNAV*. b) Plot showing average SPL for AMT users for *OBJECTNAV*. These plots clearly demonstrate some humans are better at solving the *OBJECTNAV* task than others.
Figure A.6. Screenshot of our Amazon Mechanical Turk interface for collecting PICK&PLACE demonstrations. Users are provided the agent’s first-person view of the environment and instruction such as "Pick the toy airplane and place it on the colored wood blocks". They can make the agent look around and move in the environment via keyboard controls, and can submit the task upon successful completion by clicking the ‘Submit’ button.
Figure A.7. Visualizations of learnt agent behaviors for **OBJECT NAV** and **PICK & PLACE**. Best viewed in videos at sites.google.com/view/object-search-supp.
Figure A.7. Visualizations of learnt agent behaviors for **OBJECTNAV** and **PICK&PLACE**. Best viewed in videos at ram81.github.io/projects/habitat-web.
A.7 Prior work in RL Finetuning

A.7.1 DAPG [65]

Preliminaries. Rajeswaran et al. [65] proposed DAPG, a method which incorporates demonstrations in RL, and thus quite relevant to our methodology. DAPG first pretrains a policy using behavior cloning then finetunes the policy using an augmented RL objective (shown in Eq. (Equation A.4)). DAPG proposes to use different parts of demonstrations dataset during different stages of learning for tasks involving sequence of behaviors. To do so, they add an additional term to the policy gradient objective:

\[
g_{\text{aug}} = \sum_{(s,a) \in \tau \sim \pi_\theta} \nabla_{\theta} \log \pi_\theta(a|s) A^\pi(s, a) + \sum_{(s,a) \in \tau \sim \mathcal{T}} \nabla_{\theta} \log \pi_\theta(a|s) w(s, a) \quad (A.4)
\]

Here \( \tau \sim \pi_\theta \) is a trajectory obtained by executing the current policy, \( \tau \sim \mathcal{T} \) denotes a trajectory obtained by replaying a demonstration, and \( w(s, a) \) is a weighting function to alternate between imitation and reinforcement learning. DAPG uses a heuristic weighting scheme to set \( w(s, a) \) to decay the auxiliary objective:

\[
w(s, a) = \lambda_0 \lambda_1^k \max_{(s', a') \in \tau \sim \pi_\theta} A^\pi(s', a') \forall (s, a) \quad (A.5)
\]

where \( \lambda_0 \) and \( \lambda_1 \) are hyperparameters and \( k \) is the update iteration counter. The decaying weighting term \( \lambda_1^k \) is used to avoid biasing the gradient towards the demonstrations data towards the end of training.

Implementation Details. [65] showed results of using DAPG on dexterous hand manipulation tasks for object relocation, in-hand manipulation, tool use, etc. To train the policy with behavior cloning, they use 25 demonstrations for each task gathered using the Mujoco HAPTIX system [92]. The small size of the demonstrations dataset and the observation input allows DAPG to load the demonstrations dataset in system memory which makes it
feasible to compute the augmented RL objective shown above.

**Challenges in adopting [65]’s setup.** Compared to [65], our setup uses high-dimensional visual input (256×256 RGB observations) and 77k OBJECTNAV demonstrations for training. Following DAPG’s training implementation, storing the visual inputs for 77k demonstrations in system memory would require 2TB, which is significantly higher than what is possible on today’s systems. An alternative is to leverage on-the-fly demonstration replay during RL training. However, efficiently incorporating demonstration replay with experience collection online requires solving a systems research problem. Naively switching between online experience collection using the current policy and replay demonstrations would require 2x the current experience collection time, overall hurting the training throughput.

### A.7.2 Feasibility of Off-Policy RL finetuning

There are several methods for incorporating demonstrations with off-policy RL [76, 77, 78, 79, 80]. Algorithm 1 shows the general framework of off-policy RL (finetuning) methods.

**Algorithm 1** General framework of off-policy RL algorithm

$\pi_{\theta}$: Policy, $B$: replay buffer, $N$: Rounds, $I$: Policy Update Iterations

\[
\begin{align*}
\text{for } k = 1 \text{ to } N \text{ do } \\
\text{Trajectory } \tau \leftarrow \text{Rollout } \pi_{\theta}(\cdot|s) \text{ to collect trajectory } \\
\{ (s_1, a_1, r_1, h_1), \ldots, (s_T, a_T, r_T, h_T) \} \\
B \leftarrow \{ B \} \cup \{ \tau \} \\
\pi_{\theta} \leftarrow \text{TrainPolicy}(\pi_{\theta}, B) \text{ for } I \text{ iterations}
\end{align*}
\]

Unfortunately, most of these methods use feedforward state encoders, which is ill-posed for partially observable settings. In partially observable settings, the agent requires a state representation that combines information about the state-action trajectory so far with information about the current observation, which is typically achieved using a recurrent network.

To train a recurrent policy in an off-policy setting, the full state-action trajectories need to be stored in a replay buffer to use for training, including the hidden state $h_t$ of the RNN. The policy update requires a sequence input for multiple time steps $[(s_t, a_t, r_t, h_t), \ldots, (s_{t+l}, a_{t+l}, r_{t+l}, h_{t+l})] \sim \tau$ where $l$ is sampled sequence length. Additionally, it is not obvious how the hidden state should be initialized for RNN updates when using a sampled sequence in the off-policy setting. Prior work DRQN[93] compared two training strategies to train a recurrent network from replayed experience:
**Require:** 1. **Bootstrapped Random Updates.** The episodes are sampled randomly from the replay buffer and the policy updates begin at random steps in an episode and proceed only for the unrolled timesteps. The RNN initial state is initialized to zero at the start of the update. Using randomly sampled experience better adheres to DQN’s [94] random sampling strategy, but, as a result, the RNN’s hidden state must be initialized to zero at the start of each policy update. Using zero start state allows for independent decorrelated sampling of short sequences which is important for robust optimization of neural networks. Although this can help RNN to learn to recover predictions from an initial state that mismatches with the hidden state from the collected experience but it might limit the ability of the network to rely on it’s recurrent state and exploit long term temporal correlations.

2. **Bootstrapped Sequential Updates.** The full episode replays are sampled randomly from the replay buffer and the policy updates begin at the start of the episode. The RNN hidden state is carried forward throughout the episode. Eventhough this approach avoids the problem of finding the correct initial state it still has computational issues due to varying sequence length for each episode, and algorithmic issues due to high variance of network updates due to highly correlated nature of the states in the trajectory.

Even though using bootstrapped random updates with zero start states performed well in Atari which is mostly fully observable, R2D2[81] found using this strategy prevents a RNN from learning long-term dependencies in more memory critical environments like DMLab.

[81] proposed two strategies to train recurrent policies with randomly samples sequences:

1. **Stored State.** In this strategy, the hidden state is stored at each step in the replay and use it to initialize the network at the time of policy updates. Using stored state partially remedies the issues with initial recurrent state mismatch in zero start state strategy but it suffers from ‘representational drift’ leading to ‘recurrent state staleness’, as the stored state generated by a sufficiently old network could differ significantly from a state from the current policy.

2. **Burn-in.** In this strategy the initial part of the replay sequence is used to unroll the
network and produce a start state (‘burn-in period’) and update the network on the remaining part of the sequence.

While R2D2 [81] found a combination of these strategies to be effective at mitigating the representational drift and recurrent state staleness, this increases computation and requires careful tuning of the replay sequence length \( m \) and burn-in period \( l \).

Both [81, 93] demonstrate the issues associated with using a recurrent policy in an off-policy setting and present approaches that mitigate issues to some extent. Applying these techniques for Embodied AI tasks and off-policy RL finetuning is an open research problem and requires empirical evaluation of these strategies.

A.8 Prior work in Imitation Learning

In Imitation Learning (IL), we use demonstrations of successful behavior to learn a policy that imitates the expert (demonstrator) providing these trajectories. The simplest approach to IL is behavior cloning (BC), which uses supervised learning to learn a policy to imitate the demonstrator. However, BC suffers from poor generalization to unseen states, since the training mimics the actions and not their consequences. DAgger [95] mitigates this issue by iteratively aggregating the dataset using the expert and trained policy \( \hat{\pi}_{i-1} \) to learn the policy \( \hat{\pi}_i \). Specifically, at each step \( i \), the new dataset \( D_i \) is generated by:

\[
\pi_i = \beta \pi_{exp} + (1 - \beta) \hat{\pi}_{i-1}
\]  

(A.6)

where, \( \pi_{exp} \) is a queryable expert, and \( \hat{\pi}_{i-1} \) is the trained policy at iteration \( i - 1 \). Then, we aggregate the dataset \( D \leftarrow D \cup D_i \) and train a new policy \( \hat{\pi}_i \) on the dataset \( D \). Using experience collected by the current policy to update the policy for next iteration enables DAgger [95] to mitigate the poor generalization to unseen states caused by BC. However, using DAgger [95] in our setting is not feasible as we don’t have a queryable human expert for policies being trained with human demonstrations.
Alternative approaches [96, 97, 98, 99, 100] for imitation learning are variants of inverse reinforcement learning (IRL), which learn reward function from expert demonstrations in order to train a policy. IRL methods learn a parameterized $\mathcal{R}_{\phi}(\tau)$ reward function, which models the behavior of the expert and assigns a scalar reward to a demonstration. Given the reward $r_t$, a policy $\pi_{\theta}(a_t|s_t)$ is learned to map states $s_t$ to distribution over actions $a_t$ at each time step. The goal of IRL methods is to learn a reward function such that a policy trained to maximize the discounted sum of the learned reward matches the behavior of the demonstrator.

Compared to prior works [96, 97, 98, 99, 100], our setup uses a partially-observable setting and high-dimensional visual input for training. Following training implementation from prior works, storing visual inputs of demonstrations for reward model training would require $2TB$ system memory, which is significantly higher than what is possible on today’s systems. Alternatively, efficiently replaying demonstrations during RL training with reward model learning in the loop requires solving an open systems research problem. In addition, applying these methods for tasks in a partially observable setting is an open research problem and requires empirical evaluation of these approaches.

### A.9 Training Details

#### A.9.1 Behavior Cloning

We use a distributed implementation of behavior cloning by [62] for our imitation pretraining. Each worker collects 64 frames of experience from 8 environments parallely by replaying actions from the demonstrations dataset. We then perform a policy update using supervised learning on 2 mini batches. For all of our BC experiments, we train the policy for $500M$ steps on 64 GPUs using Adam optimizer with a learning rate $1.0 \times 10^{-3}$ which is linearly decayed after each policy update. Tab. A.2 details the default hyperparameters used in all of our training runs.
Table A.2. Hyperparameters used for Imitation Learning.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of GPUs</td>
<td>64</td>
</tr>
<tr>
<td>Number of environments per GPU</td>
<td>8</td>
</tr>
<tr>
<td>Rollout length</td>
<td>64</td>
</tr>
<tr>
<td>Number of mini-batches per epoch</td>
<td>2</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$1.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0.0</td>
</tr>
<tr>
<td>Epsilon</td>
<td>$1.0 \times 10^{-5}$</td>
</tr>
<tr>
<td>DDPIL sync fraction</td>
<td>0.6</td>
</tr>
</tbody>
</table>

A.9.2 Reinforcement Learning

To train our policy using RL we use PPO with Generalized Advantage Estimation (GAE) [101]. We use a discount factor $\gamma$ of 0.99 and set GAE parameter $\tau$ to 0.95. We do not use normalized advantages. To parallelize training, we use DD-PPO with 16 workers on 16 GPUs. Each worker collects 64 frames of experience from 8 environments parallely and then performs 2 epochs of PPO update with 2 mini-batches in each epoch. For all of our experiments, we RL finetune the policy for 300M steps. Tab. A.3 details the default hyperparameters used in all of our training runs.

A.9.3 RL Finetuning using VPT

To compare with RL finetuning approach proposed in VPT [66] we implement the method in DD-PPO framework. Specifically, we initialize the critic weights to zero, replace the entropy term in PPO [83] with a KL-divergence loss between the frozen IL policy and RL policy, and decay the KL divergence loss coefficient, $\rho$, by a fixed factor after every iteration. This loss term is defined as:

$$L_{kl_{penalty}} = \rho KL(\pi^{BC}_\theta, \pi_\theta)$$  \hspace{1cm} (A.7)

where $\pi^{BC}_\theta$ is the frozen behavior cloned policy, $\pi_\theta$ is the current policy, and $\rho$ is the loss...
Table A.3. Hyperparameters used for RL finetuning.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of GPUs</td>
<td>16</td>
</tr>
<tr>
<td>Number of environments per GPU</td>
<td>8</td>
</tr>
<tr>
<td>Rollout length</td>
<td>64</td>
</tr>
<tr>
<td>PPO epochs</td>
<td>2</td>
</tr>
<tr>
<td>Number of mini-batches per epoch</td>
<td>2</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0.0</td>
</tr>
<tr>
<td>Epsilon</td>
<td>$1.0 \times 10^{-5}$</td>
</tr>
<tr>
<td>PPO clip</td>
<td>0.2</td>
</tr>
<tr>
<td>Generalized advantage estimation</td>
<td>True</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.99</td>
</tr>
<tr>
<td>$\tau$</td>
<td>0.95</td>
</tr>
<tr>
<td>Value loss coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>Max gradient norm</td>
<td>0.2</td>
</tr>
<tr>
<td>DDPPO sync fraction</td>
<td>0.6</td>
</tr>
</tbody>
</table>

weighting term. Following, VPT [66] we set $\rho$ to 0.2 at the start of training and decay it by 0.995 after each policy update. We use learning rate of $1.5 \times 10^{-5}$ without a learning rate decay for our VPT [66] finetuning experiments.

A.9.4 RL Finetuning Ablations

Table A.4. RL-finetuning ablations on HM3D val [19, 63]

<table>
<thead>
<tr>
<th>Method</th>
<th>Success (↑)</th>
<th>SPL (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) BC</td>
<td>52.0</td>
<td>20.6</td>
</tr>
<tr>
<td>2) BC→RL-FT</td>
<td>53.6 ±1.01</td>
<td>28.6 ±0.50</td>
</tr>
<tr>
<td>3) BC→RL-FT (+ Critic Learning)</td>
<td>56.7 ±0.93</td>
<td>27.7 ±0.82</td>
</tr>
<tr>
<td>4) BC→RL-FT (+ Critic Learning, Critic Decay)</td>
<td>59.4 ±0.42</td>
<td>26.9 ±0.38</td>
</tr>
<tr>
<td>5) BC→RL-FT (+ Critic Learning, Actor Warmup)</td>
<td>58.2 ±0.55</td>
<td>26.7 ±0.69</td>
</tr>
<tr>
<td>6) PIRLNav</td>
<td>61.9 ±0.47</td>
<td>27.9 ±0.56</td>
</tr>
</tbody>
</table>

For ablations presented in Sec. 4.3 of the main paper (also shown in Tab. A.4) we use a policy pretrained on $20k$ human demonstrations using BC and finetuned for $300M$ steps using hyperparameters from Tab. A.3. We try 3 learning rates ($1.5 \times 10^{-4}$, $2.5 \times 10^{-4}$, and $1.5 \times 10^{-5}$) for both BC→RL (row 2) and BC→RL (+ Critic Learning) (row 3) and we
Figure A.8. A policy pretrained on the OBJECTNAV task is used as initialization for actor weights and critic weights are initialized randomly for RL finetuning using DD-PPO. The policy performance immediately starts dropping early on during training and then recovers leading to slightly higher performance with further training.

We find BC→RL (row 2) works best with a smaller learning rate but the training performance drops significantly early on, due to the critic providing poor value estimates, and recovers later as the critic improves. See Fig. A.8. In contrast when using proposed two phase learning setup with the learning rate schedule we do not observe a significant drop in training performance.

report the results with the one that works the best. For PIRLNav we use a starting learning rate of $2.5 \times 10^{-4}$ and decay it to $1.5 \times 10^{-5}$, consistent with learning rate schedule of our best performing agent. For ablations we do not tune learning rate parameters of PIRLNav, we hypothesize tuning the parameters would help improve performance.