The Foundation Model Path to Open-World Robots

Dhruv Shah

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-166
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-166.html

August 9, 2024

The Foundation Model Path to Open-World Robots

by

Dhruv Shah

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering — Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Jitendra Malik
Professor Alexandre Bayen
Professor Dieter Fox

Summer 2024

The Foundation Model Path to Open-World Robots

Abstract

The Foundation Model Path to Open-World Robots

by

Dhruv Shah
Doctor of Philosophy in Engineering — Electrical Engineering and Computer Sciences

University of California, Berkeley
Professor Sergey Levine, Chair

Data-driven robotics has been a very effective paradigm in the last decade. Today, we can can autonomously perform dexterous tasks like folding cloths, navigate tight hallways while avoiding collisions, and control complex dynamical systems like a quadrupedal robot walking across challenging terrains using onboard observations. But they often pose fundamental limitations that prevent them from being deployed in open-world environments, either because they make strong assumptions about the structure of their environment, require large amounts of on-robot data collection, or fail to account for semantic understanding of their surroundings. Due to these limitations, data-driven robotics approaches are still limited to simple restricted settings and not accessible to a majority of practitioners and potential applications. They still need to be hand-engineered for each separate robot, in a specific environment, to solve a specific task.

This dissertation proposes an alternate vision for intelligent robots of the future, where we can have *general* machine learning models that can control any robot out of the box to perform reasonable behaviors in challenging open-world environments. Inspired by the onset of *foundation models* of language and vision, we present a recipe for training Robot Foundation Models (RFMs) from large amounts of data, collected across different environments and embodiments, that can control a wide variety of different mobile robots by only relying on egocentric vision. We also demonstrate how such an RFM can serve as a backbone for building very capable robotic systems, that can explore dense forests, or interact with humans in their environments, or utilize sources of side information such as satellite imagery or natural language.

Finally, we propose a recipe for combining RFMs, with their knowledge of the physical world, with internet foundation models of language and vision, with their image-level semantic understanding and text-based reasoning, using a novel planning framework. This enables robotic systems to leverage the strength of internet foundation models, while also being grounded in real-world affordances and act in the real-world. We hope that this is a step towards such general-purpose robotic systems that can be deployed on a wide range of robots, leverage internet-scale knowledge from pre-trained models, and serve as a foundation for diverse mobile robotic applications.

CONTENTS

II. Cross-Embodiment *Robot Foundation Models*

III. Combining Robot and Internet Foundation Models

IV. Appendices

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

This dissertation is the culmination of five enriching years at Berkeley and over a decade of explorations in the world of robotics and machine learning. While I get to take credit for the pages that follow, they would not exist without the village that nurtured and supported me through all these years.

First and foremost, I want to express my sincerest gratitude to my advisor, Sergey Levine, for his continued support and guidance throughout my PhD. Besides heavily influencing a lot of the ideas in this dissertation, Sergey taught me the art of asking the right questions, and the important skill of timely dissemination to the broader community. Sergey's most impressive quality was his ability to multitask and context-switch. Despite directing a large research group, I will never understand how Sergey had the time to think about every project, and do justice to every student. From Sergey, I learned the art of orchestrating a long-term research vision that spans multiple projects and never losing sight of the true goal, while also doing justice to each individual problem at hand. I am most grateful to Sergey for magically walking the line between steering me away from getting overly invested in the wrong questions, but also giving me the freedom to make my own mistakes and learn from them. I will never have enough words of gratitude for Sergey's impact on my life and career.

This dissertation would be incomplete without the support and mentorship of many brilliant researchers over the years. I want to thank Jitendra Malik, Alexandre Bayen, and Dieter Fox, for serving on my dissertation committee and for providing invaluable feedback at various points during my PhD. I am particularly grateful to Dieter, for many fruitful discussions during my conference presentations, and to Jitendra, for his timeless advice on picking the right problems to work on. I also want to thank Chelsea Finn, Gaurav Sukhatme, Jie Tan, Liam Paull, and Nicholas Roy, for their support during my job search and providing invaluable advice.

The seeds of this dissertation were sown during my formative years at IIT Bombay, where I was fortunate to be advised by Ajit Rajwade, on topics in signal processing and computational imaging, and Leena Vachhani, on topics in swarm robotics and stability analysis. I am particularly grateful to Leena Vachhani for sponsoring me to attend my first ever conference (ICRA 2019 at Montréal), which introduced me to the vibrant and exciting progress in robotics across the world. I have been to many conferences since, but the thrill, overwhelm, and giddy excitement of my first still remains with me. I am very grateful for the constant support and mentorship of Alankar Kotwal, who encouraged me to never give up on my explorations and inspired me to pursue a PhD. I am also grateful to Shashwat Shukla and Chinmay Talegaonkar, with whom I shared many late evenings and long nights (barely) balancing undergraduate research and coursework. Lastly, I am indebted to Sebastian Scherer at Carnegie Mellon University, Stefan Leutenegger at Imperial College London, and Donald Dansereau at University of Sydney, who generously hosted me at their labs for summer internships during this period and gave me a glimpse into life as a graduate student, eventually convinced me to pursue graduate research in robotics.

I have had the great fortune of collaborating with and learning from a number of faculty members, postdocs, students, and researchers, during the course of my PhD. I am very grateful to Abhishek Gupta, who showed me the ropes when I began my PhD and has been a great mentor and friend ever since; Greg Kahn, who taught me the importance of reliable and future-proof robot infrastructure; Ben Eysenbach, who taught me the art of crisp and fast writing; Nick Rhinehart, who introduced me to some very effective visual aids for papers: tikz, ColorBrewer, and seaborn; and to Ilya Kostrikov, who tried his best to teach me good JAX practices. Their mentorship and collaboration really set the tone in the initial years of my PhD.

Since then, I am grateful for the collaborations with Blazej Osinski, Catherine Glossop, Chelsea Finn, Dorsa Sadigh, Jonathan Yang, Kevin Black, Kyle Stachowicz, Laura Smith, Noriaki Hirose, Oier Mees and Quan Vuong. In particular, I would like to thank Noriaki Hirose and Kyle Stachowicz, with whom I explored several new research threads, only some of which made it into this dissertation. I am especially grateful for the opportunity to work with the most exceptional undergraduate students: Ajay Sridhar, Arjun Bhorkar, Hrish Leen, Michael Equi and Nitish Dashora. I had some of my best ideas when working with them, and it was an absolute honor to witness the early days of their research journeys. I wish them the best for their research endeavours in grad school and industry, and look forward to their future achievements.

During my PhD, I had the privilege of spending time at two of the leading industry labs in robotics research at Google DeepMind (then, Brain) and Meta Fundamental AI Research. At Brain, I am eternally grateful to the mentorship of Brian Ichter and Alex Toshev, who gave me the freedom to explore ideas far beyond the scope of this dissertation. This internship turned into a visiting student researcher engagement, which turned into a BAIR Commons, and became one of the longest and most fun collaborations of my PhD. Brian's intuitions about motion planning neatly connected with the rise of reasoning and planning capabilities of language models, and with him, I worked on one of the earliest attempts at incorporating language models and robot policies, which became Part III of this dissertation. I am also grateful to Jie Tan, for his mentorship during my engagements at Brain, and finally, for collaborating on scaling up parts of this dissertation at DeepMind. At FAIR, I am grateful for the mentorship of Devendra Chaplot, who taught me the importance of good simulation, benchmarks, and the *vision way* of doing robotics. I am also grateful to all the researchers and engineers at Brain and FAIR that supported me and collabored with me during these internships: Andy Zeng, Chris Paxton, Danny Driess, Dhruv Batra, Fei Xia, Igor Mordatch, Karol Hausman, Michael Chang, Mukul Khanna, Peng Xu, Pete Florence, Roozbeh Mottaghi, Sriram Yenamandra, Ted Xiao, Theophile Gervet, Tiffany Min, Tingnan Zhang, Wenlong Huang, Yao Lu and Zhuo Xu.

I had the opportunity to organize many workshops at ICRA, CoRL, RSS and NeurIPS, and am grateful to my co-organizers across the world, who I am now fortunate to call friends: Andrey, Claas, Ed, Georgia, Ishika, Kaiyu, Paula, Prayusha, Mohit, Oier, Sid, Suraj, Ted, Victoria and Yuqing. I am particularly grateful to Ted, Suraj and Oier, for their support in enabling three consecutive editions of the LangRob workshop at CoRL, which has evolved into being my favorite day of my favorite conference every year. I also grateful to friends and colleagues the broader robotics and machine learning communities, too many to name, who I look forward to meet at upcoming conferences for many years to follow.

I am very grateful to all the members of the RAIL lab and the broader BAIR and EECS communities for creating an enriching and supportive atmosphere in Berkeley Way West. COVID-19 hit us right as we moved into this beautiful building, and a lot of my PhD social life was remote, but I am very thankful for all the people that made BWW8 an engaging atmosphere. Between lazy Fridays playing ping pong or whiling away in the kitchen, catching the most beautiful sunset every evening over the San Francisco skyline in the distance, and the frequent deadline all-nighters marked by watching sunrise over the Berkeley hills, BWW became my second home and I am grateful to all the members of the BAIR community that made it fun. I had the privilege of sharing our lab space with some great colleagues, that made life fun over the past five years: Abhishek, Amrith, Amy, Anusha, Aurick, Avi, Aviral, Ashvin, Blazej, Catherine, Charlie, Coline, Colin, Dibya, Dinesh, Frederik, Glen, Greg, Homer, Ilya, James, Jason, JD, Jianlan, Joey, Justin, Karl, Kate, Katie, Kelvin, Kevin, Kevin, Kuan, Kyle, Laura, Manan, Marvin, Marwa, Michael C, Michael J, Michal, Mitsuhiko, Natasha, Nick, Noriaki, Oier, Oleg, Paul, Philip, Pranav, Roberto, Rowan, Seohong, Sid, Simon, Sudeep, Tim, Vikash, Vitchyr, William, Yifei, Young. I am also grateful to Ashish, Antonio, Ilija, Ryan, Sasha, Shiry, and many others at BAIR who were always around for bouncing ideas and for all the fruitful discussions over the years. BWW would have burned down multiple times if not for the amazing Angie Abbatecola, Roxana Infante, and Ami Katagiri, who made sure everything was in shape and there was always food after the seminars. I am also very grateful to Jean Nguyen, Shirley Salanio, and Judy Smithson, for their support during the various phases of my PhD, and making sure I can actually graduate in time.

Outside of work, Berkeley became a truly special place thanks to my amazing set of friends, at Berkeley and otherwise. I am very grateful to all the lovely people from the 2019 cohort that made the transition into grad school smooth and enjoyable: Alok, Ilija, Kartik, Kathy, Lucy, Medhini, Orr, Rudy, Suzie, Tim, Yahav and Yuqing. Whether it was late night karaoke, hiking, Thanksgiving bar hopping, or just growing old together, I am glad I met these amazing bunch of friends and colleagues. Berkeley life would not have been half as fun without my partners in *crime*: Kathy and Yuqing. I am also particularly grateful to the Efros-Malik group, for housing me in my first year and for organizing the best hiking and camping trips, and to AUTOLab, for incorporating me as a pseudo-labmate at all the fun robotics conferences we've attended together. I am grateful to Aagam and Tarini, for entertaining all my crazy camping ideas with the enthusiasm of *vultures*, to my roommates Asish and Deepto, for making sure I was never homesick, and to all other friends that made life in the Bay fun and homely: Arunabh, Bhabesh, Chinmay, Devansh, Karan, Pallavi, Shankar, Shubham and Upadhi. I am also very grateful for my closest friends in India and around the world, who made sure I always felt close and always made time when I felt homesick: Devyani, Neeraj, Kanush, Anmol, Sanket and Tanya.

This dissertation would not have existed without Medini, who kept me sane every day, through thick and thin. She gave me a life outside of work to look forward to — whether its learning how to drive, ski, climb boulders, ice skate, or buying a kayak, or traveling around the world, or helping me as a subject for robot experiments. . . . I am also very grateful for my wonderful family, who has always been there for me throughout the years before and during my PhD: Sweety, Viral, Viha and Dhruvi in CA, Raj, Bhumi and Rihaan in OH, Shlesha and Jalpen in NJ, and Het, Adit, Alisha and Riddhi in India. Finally, I want to thank my parents Ilesh and Shital for their love and support throughout my life. I can't imagine the courage it takes to send your son away halfway across the globe and only get to meet every other year, and I cherish the few minutes a day I get to catch up with them and escape from my busy life. I would not be half the person I am today without their sacrifices and nurturing since childhood, and I extremely fortunate for their support.

1

INTRODUCTION

When humans visit a new city, we have no trouble navigating the streets, asking for directions to landmarks, and making a mental map of the neighborhood. We draw from our previous experiences and common sense to recognize patterns and fill in the blanks, such as ″city blocks are typically rectangular″ and ″exit signs lead to doors″. However, the majority of robotic systems today lack such awareness and reasoning capabilities in unseen, unstructured *in-the-wild* environments — they either (i) model too much (geometry) and have limitations in terms of scalability, and adaptability, or (ii) model too little and try to learn everything, with constraints on efficiency, generalization, and robustness. Recent advancements in artifical intelligence and large-scale deep learning have shown that such reasoning is indeed present in neural networks that model text and visual information. So, why are we yet to see autonomous robots that can efficiently explore city-scale environments, use common-sense reasoning, and co-exist with humans *in-the-wild*?

The key challenge in building such a *general-purpose* robot lies in the lack of algorithms and systems that can model diverse robot behaviors in arbitrary homes, streets or forests. These behaviors can be arbitrarily complex and impossible to reconstruct or simulate procedurally. As a result, neither classical methods relying on dense mapping and geometry, nor modern end-to-end learning methods trained in simulation will get us towards this goal. We posit that the answer lies in building robotic systems that can learn from *shared* real-world experiences, i.e. data collected from real-world deployment of robots across the world, with different sensors and capabilities, and solving different tasks. This dissertation focuses on harnessing the power of "cross-embodiment" robot learning to address the aforementioned challenges, and connecting them to the rigorous foundations of planning and search, with the ultimate goal of building robust systems capable of real-world deployment in complex environments. This paradigm can enable broad generalization that enables entirely new robots to be deployed *in-the-wild*, as well as efficient adaptation to a variety of downstream tasks in a data-efficient manner, much like the successes of large pre-trained models in computer vision and language modeling. When developing robotic systems under this paradigm, it is important to consider two fundamental questions:

1. **How can we learn robust behavior** from offline datasets of robots deployed in challenging environments?

2. **How can we make these behaviors generalize** to challenging new environments, tasks, and robotic embodiments, and build a *robot foundation model*?

3. **How can combine such a model with internet foundation models** to benefit from internet-scale knowledge and high-level planning abilities?

This dissertation presents a recipe for building robot foundation models, that can enable autonomous robots to learn from their experience as well as other sources of information on the internet, and how these robot-specific models can be combined with foundation models trained on internet-scale data. We study this primarily in the context of autonomous navigation, a fundamental component of intelligent robotic systems operating beyond the laboratory, and truly *in-the-wild*. Our key insight is that the combination of large-scale robot learning from cross-embodiment data and planning can enable deployable general-purpose robots. This dissertation is divided into three parts, corresponding to setting up the visual navigation problem in a data-driven framework, designing and training a *robot foundation model* for visual navigation, and finally, combining this robot-specific model with internet foundation models of text and vision.

For the sake of completeness and disambiguation, we define a *robot foundation model* in the context of this dissertation as follows.

Definition 1 (Robot Foundation Model) *A machine learning model trained with minimal external supervision, that can be:*

- ***deployed zero-shot*** *in completely different (useful) settings*
  e.g., different sensors, robots, environments etc.

- ***adapted*** *to a downstream task of choice*
  e.g., different objectives, preferences/rewards, modality of goal specification, behaviors etc.

ORGANIZATION

This dissertation is organized in three parts, aimed at answering the questions posed above. We make the following contributions:

I. **Learning Long-Range Navigation from Data:** We formulate the visual navigation problem into a mapless data-driven paradigm. We posit that robust navigation in challenging, real-world environments requires both the ability to learn skills from past experience of the robot, as well as an explicit memory for planning and search. We also evaluate the suitability of offline reinforcement learning and behavior cloning for training robust, real-world skills.

   - In Chapter 2, we lay the groundwork for a novel learning-based navigation system that can learn entirely from offline data and perform long-range navigation via planning. This work appeared previously as Shah et al. [241].

- In Chapter 3, we build upon our system by enabling it to perform autonomous exploration by training an exploration prior using a learned latent variable model. This work appeared previously as Shah et al. [238].
- In Chapter 4, we extend this system so it can utilize geographic side information as a planning heuristic to achieve kilometer-scale navigation. This work appeared previously as Shah et al. [232].
- In Chapter 5, we study the viability of offline reinforcement learning for training robot skills from data, and demonstrate a capable navigation system that can optimize a user-specific reward function in the real world. This work appeared previously as Shah et al. [237].

II. **Cross-Embodiment Robot Foundation Models:** We propose cross-embodiment learning as a means to train goal-reaching navigation policies across several environments and robots. We explore the design space for such a model, the pre-training paradigm, as well as possible ways it can be adapted for downstream applications.

- In Chapter 6, we explore how the experiential learning paradigm can be extended to learn navigation policies *across different robotic embodiments* by careful data curation and architecture design. This work appeared previously as Shah et al. [234].
- In Chapter 7, we design the first ever *robot foundation model* based on cross-embodiment learning. The ViNT model aims to bring the success of pre-trained models to robotics, serving as a powerful backbone for training downstream policies via prompt-tuning and full-model finetuning. This work appeared previously as Shah et al. [242].
- In Chapter 8, we propose an alternate architecture for cross-embodiment robot learning at scale that uses a goal-conditioned diffusion policy to learn complex, multimodal behaviors. This work appeared previously as Sridhar et al. [254].

III. **Combining Robot and Internet Foundation Models:** We propose a novel planning framework to combine the text-based reasoning capabilities of large language models, semantic understanding and visual grounding of vision models, and the physical grounding of robot foundation models to solve real-world instruction following tasks. We further combine this idea with the heuristic-based planning framework from Chapter 4 to propose a novel way to incorporate language model *suggestions* during planning.

- In Chapter 9, we design the first ever real-world instruction following robotic system that was can be deployed zero-shot in novel environments without any additional training. LM-Nav combines powerful robot foundation models

(ViNG, GNM) with internet foundation models (GPT-3, CLIP) to obtain a powerful and versatile robotic navigation system. This work appeared previously as Shah et al. [235].

- In Chapter 10, we study how we can use the inherent knowledge stored in large language models as a planning heuristic for solving long-range reasoning tasks. This work appeared previously as Shah et al. [236].

We conclude with a discussion of current approaches and promising future directions in developing truly autonomous robots that can be deployed in open-world environments.

Part I

LEARNING LONG-RANGE NAVIGATION FROM DATA

2

LEARNING OPEN-WORLD NAVIGATION WITH VISUAL GOALS

Synopsis

We propose a learning-based navigation system for reaching visually indicated goals in a previously seen environment, and demonstrate this system on a real mobile robot platform. By combining a learned policy with a topological graph constructed out of previously observed data, our system can determine how to reach this visually indicated goal even in the presence of variable appearance and lighting. Three key insights, waypoint proposal, graph pruning and negative mining, enable our method to learn to navigate in real-world environments using only offline data, a setting where prior methods struggle. We instantiate our method on a real outdoor ground robot and show that our system, which we call ViNG, outperforms previously-proposed methods for goal-conditioned reinforcement learning. We also study how ViNG generalizes to unseen environments and evaluate its ability to adapt to such an environment with growing experience. Finally, we demonstrate ViNG on a number of real-world applications, such as last-mile delivery and warehouse inspection.

2.1 INTRODUCTION

Visual navigation in complex environments poses several challenges: (i) difficulty in faithfully modeling the complex dynamics and nuanced environmental interactions; (ii) reacting to high-dimensional observations; (iii) cost and safety constraints on collecting data, requiring learning from previously collected (i.e., “offline”) experience; and (iv) generalizing effectively across different settings and environments. Planning algorithms achieve many of these desiderata, but their efficacy depends on having the right representation of the task; it remains unclear how to apply many planning algorithms to tasks with image-based observations. On the other hand, humans seemingly have little difficulty navigating complex environments from first-person observations, without GPS

Shah, Eysenbach, Kahn, Rhinehart, Levine, ”ViNG: Learning Open-World Navigation with Visual Goals”, in IEEE International Conference on Robotics and Automation (ICRA) 2021
Project website: sites.google.com/view/ving-robot

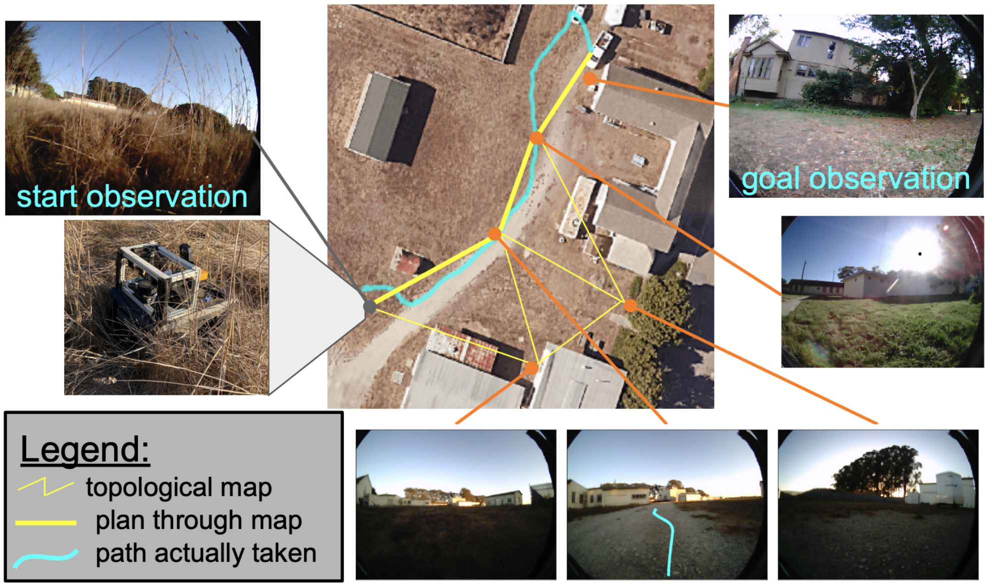


Figure 1: ViNG builds and plans over a *learned* topological graph consisting of previously seen egocentric images, and uses a learned controller to execute the path to a visually indicated goal. Unlike prior work, our method uses purely offline experience and does not require a simulator or online data collection. Note that the graph constructed by our algorithm is not geometric and nodes are *not* associated with coordinates in the world, but only with *image observations* – the top-down satellite image is provided only for visualization and is not available to our method.

or maps, if they have seen the environment before. Humans and animals are known to use "mental maps" that rely on landmarks and other cues [193, 83, 73], and rely heavily on learning. Further, in the absence of spatial positional information (e.g., GPS or maps), specification of a navigational goal itself becomes challenging, since locational goals require the robot to be able to compare its location to the target.

In this chapter, we study learning-based methods for navigation that can similarly utilize graph-structured "mental maps" that are non-geometric in nature, and can enable a robot to navigate in the real-world. We use a natural and intuitive mechanism for specifying goals – where the user provides the robot with a picture of the desired destination. Inspired by humans navigating toward previously seen landmarks, our goal is to enable the robot to navigate to a visually indicated goal. Crucially, such a goal specification scheme does not presume any prior geometric knowledge of the scene, while still providing enough information for the robot to perform the task. Fig. 1 shows an example of such a task.

Towards satisfying these requirements, we present a fully autonomous, self-supervised mobile robot platform for visual goal-reaching in outdoor, unstructured environments which we call ViNG — **Vi**sual **N**avigation with **G**oals. Our approach combines the strengths of dynamical distance learning and graph search. We first learn a function that predicts the *dynamical* distance between pairs of observations, estimating how many time steps are needed to transition between them. We then use this learned dynamical distance to embed past observations into a topological graph, and plan over this graph. This process makes no geometric assumptions about the environment: reachability is determined entirely by learning from data. Unlike pure planning-based approaches, our method scales to high-dimensional observations and hard-to-model dynamics, and does not assume access to any ground-truth spatial information. Unlike pure learning-based approaches, our method effectively learns from offline experience and reasons over long horizons. Unlike prior methods that combine planning and learning, ViNG learns from offline, real-world data, and does not require a simulator or online data collection.

The primary contribution of this work is a self-supervised robotic system, ViNG, that can efficiently learn goal-directed navigation behaviors in open-world environments without access to spatial maps from an offline pool of data, including randomly collected trajectories. Three key ideas, waypoint proposal, graph pruning and negative mining, differentiate our method from prior work and are critical to the success of our method in this offline setting. ViNG can learn to navigate to an arbitrary user-specified visual goal in a variety of open-world settings, including urban, grassy, and rocky terrain, learning only from offline data. Our experiments show that ViNG learns goal-conditioned behaviors that can effectively plan over long horizons. We show that ViNG outperforms several competitive offline RL and geometric baselines. Further, we show that the learned behaviors transfer to novel environments using as little as 20 minutes of data from the environment and that ViNG can adapt in such novel environments as it gathers more data, resulting in an autonomous, self-improving system. Lastly, we demonstrate two real-world applications enabled by ViNG in dense, urban neighborhoods – last-mile delivery of food or mail, and autonomous inspection of warehouses.

2.2 RELATED WORK

Prior work has studied vision-based mobile robot navigation in many real-world settings, including indoor and outdoor navigation [216, 265, 19], autonomous driving [267, 271], and navigation in extra-terrestrial and underwater environments [141, 48]. The combination of mapping [266] and path planning [149] has been a cornerstone for a number of effective systems [52, 246, 78] and underlies several state-of-the-art navigation systems [13, 2]. Many prior methods make restrictive assumptions, such as access to LIDAR or other structured sensor information and accurate localization, which can limit their suitability for deployment in unstructured environments [49]. Further, prior work often assumes that geometric traversability is faithfully indicated through observations and not misled by (say) non-obstacles such as tall grass [76]. Learning-based systems lift some of these assumptions and can use learned models to perform perception [34, 277], planning [123, 146, 91], or both [154]. In practice, learning temporally extended long-horizon skills with either reinforcement learning (RL) or imitation learning (IL) remains difficult [217, 63].

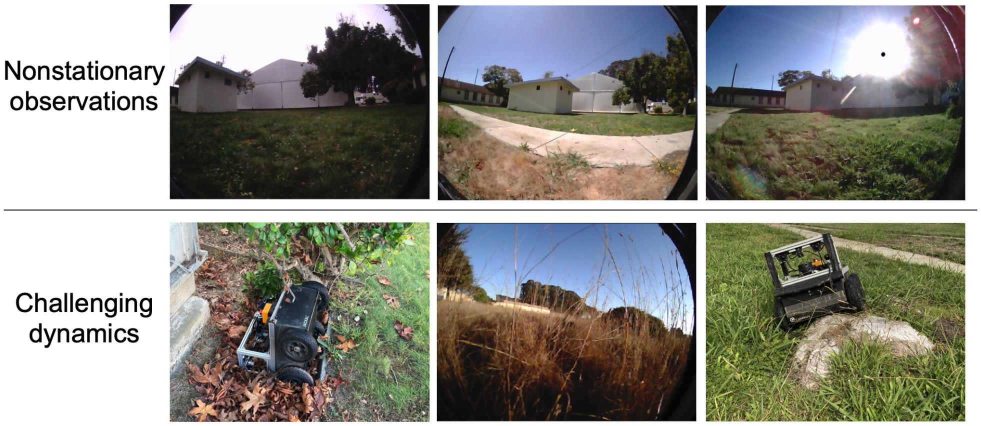


Figure 2: Challenges with Real-World Navigation: *(Top)* Three observations taken from *exactly the same position* at different times of day exhibit large differences. *(Bottom)* While tall grass and inclined rocks are traversable, a hole filled with dry leaves is not. These examples highlight the challenges with geometric reasoning about traversability.

Recent methods address limitations of the above approaches by combining planning and learning [65, 224, 28, 43, 69, 179]. These methods use learning (i.e., approximate dynamic programming) to solve short-horizon tasks and plan (i.e., use exact dynamic programming) over non-metric topological graphs [178, 177] to reason over longer horizons. This general approach simultaneously avoids the need for (1) high-fidelity map building and (2) learning temporally-extended behaviors from scratch. However, prior instantiations of this recipe make assumptions that limit their applicability to real-world settings: assuming access to an exact simulation replica of the environment [75, 179], assuming simplified action spaces [65, 224, 28], or requiring online data collection [65, 28]. Our experiments in Section 2.5 demonstrate that prior methods fail when they are not allowed to collect new experience in a simulator or the real-world.

Our method, ViNG, builds on these prior approaches by adding two key ideas: graph pruning and negative sampling. These additional ingredients allow ViNG to lift assumptions made by prior methods: it does not assume access to a simulator, and does not require interactive access to an environment; it is trained using offline, real-world data; and it operates directly on high-dimensional images and predicts continuous actions for the robot. To the best of our knowledge, ViNG is the first system demonstrated on a real-world ground robot that can learn from offline data to reach visually indicated navigational goals over long time horizons without simulated training or hand-designed localization and mapping systems.

2.3 Problem Statement and System Overview

We consider the problem of goal-directed visual navigation: a robot is tasked with navigating to a goal location G given an image observation o_G taken at G. In addition to navigating to the goal, the robot also needs to recognize *when* it has reached the goal, signaling that the task has been completed. The robot does not have a spatial map of the environment, but we assume that it has access to a small number of trajectories that it has collected previously. This data will be used to construct a graph over the environment using a learned distance and reachability function. We make no assumptions on the nature of the trajectories: they may be obtained by human teleoperation, self-exploration, or a result of a random walk. Each trajectory is a dense sequence of observations $o_1, o_2, \ldots, o_n$ recorded by its on-board camera. Since the robot only observes the world from a single on-board camera and does not run any state estimation, our system operates in a partially observed setting. Our system commands continuous linear and angular velocities.

2.3.1 *Mobile Robot Platform*

We implement ViNG on a Clearpath Jackal UGV platform – a small, fast, weatherproof outdoor ground robot ideal for navigating in both urban and off-road environments (see Fig. 1 and 2). The default sensor suite consists of a 6-DoF IMU, a GPS unit for approximate global position estimates, and wheel encoders to estimate local odometry. In addition, we added a forward-facing 170° field-of-view camera and an RPLIDAR 2D laser scanner. Inside the Jackal is an NVIDIA Jetson TX2 computer. While the robot carries a GPS and laser scanner, we use these sensors solely as a safety mechanism during data collection. Our method solely operates using images taken from the onboard camera.

2.3.2 *Data Collection & Labeling*

ViNG can learn navigational behaviors from previously-collected, off-policy data – a desideratum of real-world robots. To demonstrate this capability, we run our core experiments using data exclusively from prior work [124]; we also collect a limited amount of additional data for our environment generalization experiments using the same self-supervised data collection strategy. The prior data was collected more than 10 months prior to the experiments in this paper (see Fig. 2 *(top)*), and exhibits significant differences in appearance, lighting, time of year, and time of day as compared to the evaluation setting. This underscores the ability of ViNG to utilize offline data from diverse sources.

Algorithm 1 Training ViNG

```
**Input** transitions $\{\tau^{(k)} = (o_1^{(k)}, a_1^{(k)}, o_2^{(k)}, a_2^{(k)}, \cdots)\}_{k=1,\cdots}$
$\mathbb{D}_+ \leftarrow \{(o_i^{(k)}, o_j^{(k)}, d = \min(j - i, d_{\max}))\}_{i \leq j, k=1,\cdots}$
$\mathbb{D}_- \leftarrow \{(o_i^{(k)}, o_j^{(\ell)}, d = d_{\max})\}_{i,j,k \neq \ell}$
Initialize $\mathcal{T}(o_i, o_j)$ and $\mathcal{P}(o_i, o_j)$
**while** not converged **do**
    $\mathcal{B}_+ \sim \mathbb{D}_+, \mathcal{B}_- \sim \mathbb{D}_-$                ▷ Sample batch.
    $\mathcal{T} \leftarrow$ UpdateDistanceFn$(\mathcal{T}; \mathcal{B}_+ \cup \mathcal{B}_-)$
    get relative pose: $\mathbb{D}_+ \leftarrow \{((o_i^{(k)}, o_j^{(k)}, d_{ij}, p_{ij}\}$
    $\mathcal{P} \leftarrow$ UpdateRelativePoseFn$(\mathcal{P}; \mathcal{B}_+ \cup \mathcal{B}_-)$
**end while**
**return** traversability function $\mathcal{T}$, relative pose function $\mathcal{P}$
```

2.4 VISUAL NAVIGATION WITH GOALS

We approach the problem of visual goal-conditioned navigation by combining non-metric maps and learned, image-based, goal-conditioned policies. We describe our method in two stages: (i) training two learned functions and (ii) deploying the system, which entails using the learned functions together with past experience to execute goal-directed behavior.

During *training*, we use previously collected experience to learn an environment-independent traversability function $\mathcal{T}$, as well as a relative pose predictor, $\mathcal{P}$. During *deployment*, the robot builds a topological graph of its environment: a directed graph with vertices as observations and edges encoding traversability and proximity. At each time step t, the robot localizes its current and goal observations (o_t, o_G) in the graph and follows the best path to G, as determined by a graph search algorithm that outputs the next waypoint for the controller. To close the loop, we need a goal-conditioned controller that takes the current and goal observations, and outputs an action a. The controller progressively follows the path directed by the planner until it reaches G.

While the general recipe of ViNG is similar to prior work [224, 179, 65], our experiments demonstrate that two key technical insights contribute to significantly improved performance in the real-world setting: *graph pruning* (Sec. 2.4.2) and *negative mining* (Sec. 2.4.1). Our comparisons to prior methods in Section 2.5 and ablation studies in Section 2.5.4 demonstrate these novel improvements enable ViNG to learn goal-conditioned policies entirely from offline data, avoiding the need for simulators and online sampling, while prior methods struggle to attain good performance, particularly for long-horizon goals.

2.4.1 Learning Dynamical Distances

We aim to learn a traversability function $\mathcal{T}(o_i, o_j) \in \mathbb{R}^+$ that reflects whether *any* controller can successfully navigate between observations o_i and o_j. More precisely, we will learn to predict the estimated number of time steps required by a controller to navigate from one observation to another. This function must encapsulate knowledge of physics beyond just geometry. For example, tall grass and bushes might appear visually similar, but grass is compliant and traversable whereas bushes are not. We explored two methods for learning this traversability function: (1) supervised learning and (2) temporal difference learning [259, 122]. To learn the distance function via supervised learning, we create a dataset $\mathbb{D}_+$ of observation pairs (o_i, o_j) taken from the same trajectory and regress to the number of timesteps $d_{ij} = j - i$ elapsed between these observations. The distance predicted by this approach corresponds to the estimated number of time steps required by the behavior policy (that which collected the experience) when navigating between two observations. Thus, this approach is simple but may overestimate the true shortest path distances.

The second approach to learning the distance function is via temporal difference learning [122]. This approach uses the same experience as before. While this approach adds additional complexity, in theory it converges to the shortest path distance. In our experiments, we found little difference between these two approaches (see Table 2), but expect that the temporal difference learning approach would be important when moving to settings where the shortest path distance is much shorter than a random walk distance.

Negative Mining (Key Idea 1)

In our experiments, we found that training the distance function using only observation pairs from the same trajectory performed poorly. We hypothesize that the root cause was distribution shift: when building the topological graph we must evaluate the distance function on observation pairs collected from different trajectories, possible from different times of day. To mitigate this problem, we augment the dataset by adding a new dataset $\mathbb{D}_-$ obtained by sampling observations from different trajectories, labeled as $d_{\max}$. We find this augmentation, hereby referred to as *negative sampling*, to be critical in the successful training and evaluation of $\mathcal{T}$ in our experiments, offering significant improvements over prior methods.

2.4.2 The Topological Graph

We build a topological graph $\mathcal{M}$ using the learned distance function together with a collection of previously-observed observations $\{o_t\}$. Each *node* in the graph corresponds to one of these observations. We add weighted *edges* between every node, using weights predicted by the distance function $\mathcal{T}$.

Algorithm 2 Deploying ViNG

```
Input current image $o_t$, goal image $o_G$, and topological graph $\mathcal{M}$.
Add $o_t, o_G$ to the map $\mathcal{M}$ using distances from $\mathcal{T}$.
$o_{w_1}, o_{w_2}, \cdots \leftarrow$ Dijkstra(start $= o_t$, goal $= o_G, \mathcal{M}$)
Estimate relative pose of first waypoint: $\Delta p \leftarrow \mathcal{P}(o_t, o_{w_1})$
$u_t \leftarrow$ PD-Controller($\Delta p$)
return control $u_t$
```

Graph Pruning (Key Idea 2)

As the robot gathers more experience, maintaining a dense graph of traversability across all observation nodes becomes redundant and infeasible, as the graph size grows quadratically. For our experiments, we sparsify trajectories by thresholding the edges that get added to the graph: edges that are easily traversable $\left(\mathcal{T}(o_i, o_j) < \delta_{\text{sparsify}}\right)$ are not added to the graph, since the controller can traverse those edges with high probability.

Planning with the Graph

We localize the current observation o_t and goal observation o_G in the graph, adding direct edges (weighed by their traversability) to their corresponding "most-traversable" neighbors. We use the weighted Dijkstra algorithm to compute the shortest path to goal, and the immediate next node in the planned path is then handed over to the controller.

2.4.3 *Designing the Controller*

After the planner predicts a waypoint observation, the controller must output an action that takes the agent towards that waypoint. The main challenge in navigating to this waypoint is that both the current state and waypoint are represented as high-dimensional observations (e.g., images). To address this challenge, we learn a relative position predictor $\mathcal{P}$ that takes as input two observations and predicts the relative pose between these observations. We learn this relative pose predictor via supervised learning: for pairs of observations (o_i, o_j) that occur nearby within the collected trajectories, we estimate the relative pose Δp_{ij} using onboard odometry and use this relative pose as the label for learning.

The complete controller works as follows. Given the current observation and waypoint observation, we use the relative pose predictor to estimate the relative pose of the waypoint relative to the robot's current position. The robot then uses odometry and a simple PD controller to steer toward this waypoint. We compare against alternative controllers in Section 2.5.4.

2.4.4 *Implementation Details*

Inputs to the traversability function $\mathcal{T}$ and relative pose predictor $\mathcal{P}$ are pairs of observations of the environment, represented by a stack of two consecutive RGB images obtained from the onboard camera at a resolution of 160×120 pixels. $\mathcal{T}$ comprises a MobileNet encoder [104] followed by three densely connected layers to project the $1024-$dimensional latents to 50 class labels. $\mathcal{P}$ has a similar architecture as $\mathcal{T}$, comprising of a MobileNet encoder followed by three densely connected layers projecting the $1024-$dimensional latents to 3 outputs for waypoints: $\{\Delta x, \Delta y\}$. Both $\mathcal{T}$ and $\mathcal{P}$ use the same encoder.

We train the traversability function on $\mathbb{D}_+ \cup \mathbb{D}_-$, discretizing the timesteps d_{ij} into bins $\{1, \cdots, d_{\max} = 50\}$ and minimizing the cross entropy loss. The relative pose predictor $\mathcal{P}$ is trained on $\mathbb{D}_+$ to minimize the ℓ_2 regression loss. We use a batch size of 128 and perform gradient updates using the Adam optimizer [133] with learning rate $\lambda = 10^{-4}$. Algorithms 1 and 2 summarize our approach in the training and deployment stages, respectively.

2.5 EXPERIMENTS

We designed our experiments to answer three questions:

Q1. How does ViNG compare to prior methods for the task of goal-conditioned visual navigation from offline data?

Q2. Does ViNG generalize to novel environments? Can it adapt on the fly?

Q3. What are the alternate design choices for the controller and how do they compare against our choice in Section 2.4.3?

2.5.1 *Goal-Conditioned Visual Navigation from Offline Data*

We perform our evaluation in a real-world outdoor environment consisting of urban and off-road terrain. We train on 40 hours of data that was gathered in prior work [124] over 10 months prior to the experiments in this paper. The data shows significant variation in appearance due to seasonal changes (see Fig. 2); learning navigational affordances and traversability would require the algorithms to discard the irrelevant modes of variance (e.g., appearance) and establish correspondence across seasons and times of day.

Since this evaluation takes place in the real world, we do not have the luxury of training online RL policies or transfer from simulation. We evaluate ViNG against four baselines:

SPTM: a dense topological graph combined with a controller that maps observation pairs to motor commands, trained via supervised learning [224]

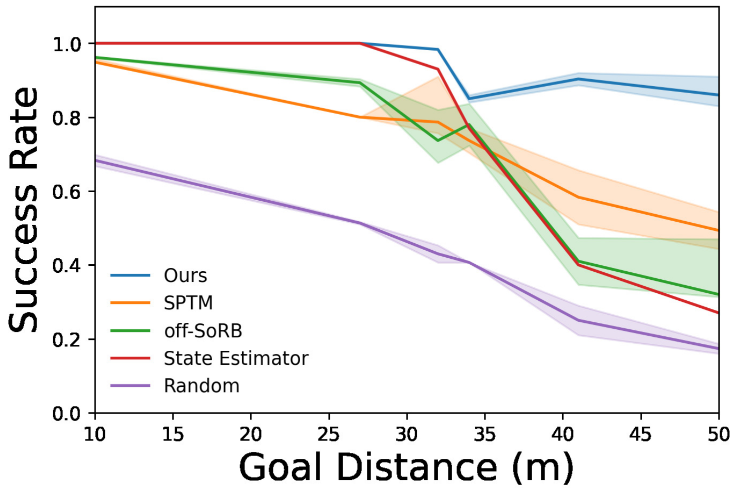


Figure 3: Real-World Navigation: While all non-random methods successfully reach nearby goals, only ViNG reaches goals over 40 meters away. Here, success rate is defined as the average over portion of the expert trajectory to goal that each run successfully completes.

off-SoRB: an offline variant of SoRB that uses a topological graph and offline RL to learn a distributional Q-function [65]

State Estimator: a naïve baseline that uses a state estimator network that regresses observations to ground-truth state (x, y, θ), followed by a position controller; note that this baseline has access to true position (from GPS), which is not available to our method

Random: a random walk, as described in Section 2.4.3

While there have been other successful instantiations of methods combining planning and learning, they make some limiting assumptions that make them difficult to apply to our problem setting. *LSTN* [179] uses a photorealistic simulator to train its distance and action models, using $\sim$ 1.5M samples, while *PRM-RL* [75] uses a 3D kinematic simulator simulation replica to train a reactive controller, coupled with physical rollouts in the real world to build a PRM. ViNG does not assume access to any simulator, and learns directly from offline real data.

Towards answering Q1, we evaluate the goal-reaching performance of ViNG. We select 6 {start, goal} image pairs in the original urban environment and compare the goal reaching performance of each method (avg. of 3 trials). We report the success metric as the average over portion of the expert trajectory to goal that each run successfully completes.

As shown in Fig. 3, ViNG performs well on all tasks, achieving a success rate of 86% on even the most challenging tasks. As expected, the random baseline, which ignores

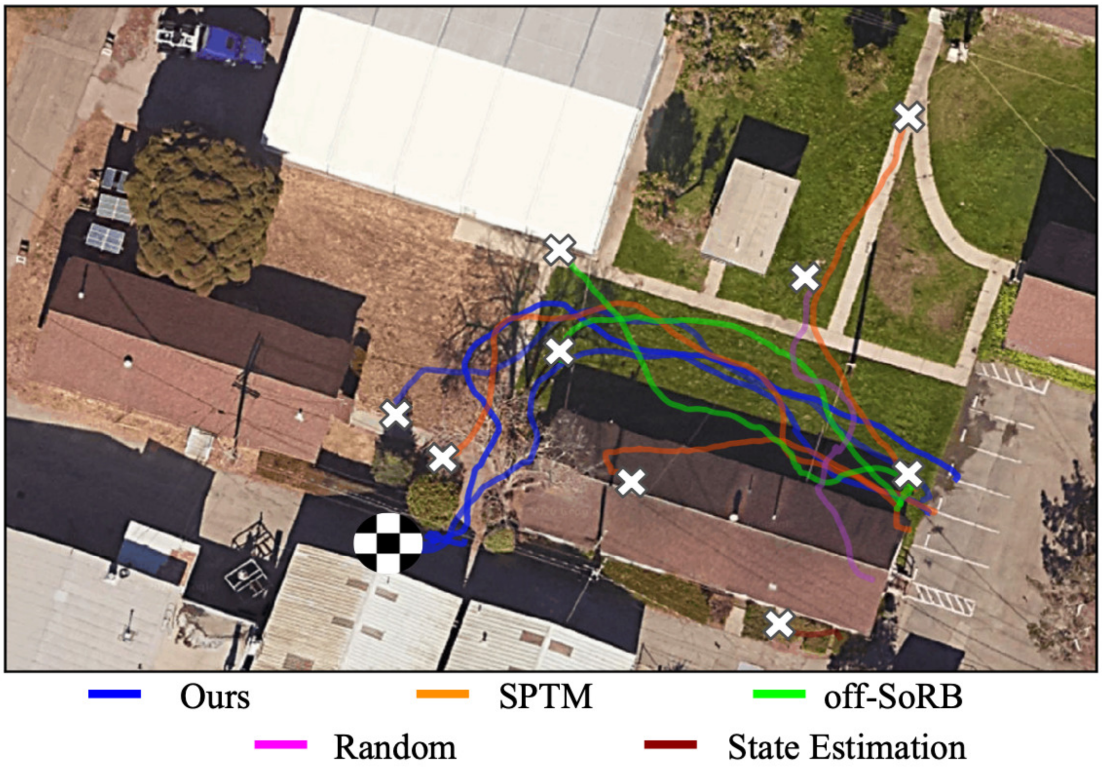


Figure 4: Qualitative Results in the urban **Environment**: Each approach was directed to a visual goal $\sim$ 50m away (marked by checkerboard circle) – with 3 runs per approach. ViNG is the only approach that is consistently able to reach the goal while avoiding collisions or getting stuck.

the goal, fails to reach most goals. The state estimator baseline performs a bit better, but struggles to reach more distant goals because it is not reactive, and hence cannot take actions to avoid collisions. Off-SoRB performs well on nearby goals, but as the goals get increasingly difficult to reach, it is unable to follow the planned trajectory. Visualizing the topological graph built by SoRB uncovers many disconnected components, resulting in no path to goal. We hypothesize that this is attributed to the difficulty in training Q-functions from offline data. SPTM, which uses supervised learning instead of Q-learning, is effective at solving the task on shorter horizons and outperforms off-SoRB on longer horizons. However, ViNG still performs substantially better on all goal distances, especially those over 30 meters. We attribute these improvements to the additional negative sampling and graph pruning techniques discussed in Section 2.4. We visualize trajectories in Fig. 4.

2.5.2 *Generalization and Adaptation*

The experiments in the previous section evaluate navigation to new goals in a previously seen environment. In this section, we additionally evaluate how quickly ViNG can adapt to an entirely new, unseen environment, by constructing a new graph and finetuning the models. We use the four settings shown in Fig. 5, all of which are distinct from the setting used in our main experiments (Sec. 2.5.1). In each new environment, a human controlled the robot to provide initial exploration data. After this initial data collection, the robot collected experience *autonomously*: it randomly sampled a previously-observed image as thegoal and used ViNG to attempt to reach this goal. After each episode, we used all experience from the new environment (both the expert trajectories and the self-collected trajectories) to finetune $\mathcal{T}$ and $\mathcal{P}$. We refer to this approach to generalization as ViNG -Finetune.

In Fig. 5 we visualize trajectories after 60 min of data collection in the new environment and observe that the robot successfully reaches the goal in most cases. We emphasize that these environments are considerably different from those used in Sec. 2.5.1, on which our models were initially trained. To illustrate the learning dynamics in this generalization setting, we plot self-collected rollouts after 0 minutes, 20 minutes, and 60 minutes of practice in the new environments. As shown in Fig. 6, the robot's performance in the new domain gets progressively better with more (autonomous) practice; after 60 minutes it succeeds in reaching the goal in all three attempts.

Table 1 summarizes the success rate on the generalization task of our method and two alternative versions of ViNG. ViNG-Source directly uses the traversability function and relative pose function trained in the source domain (Sec. 2.5.1), without incorporating any experience from the new environment. In contrast ViNG-Target learns these same models using only experience from the new "target" domain, without leveraging any of the previously-collected experience. ViNG-Finetune outperforms these baselines, highlighting the importance of combining old and new experience. As an additional baseline, we take the SPTM model from Sec. 2.5.1 and finetune it on experience from the new domain. We observe that ViNG -Finetune also generalizes better than SPTM-Finetune, We hypothesize that ViNG generalizes better than SPTM because of the additional hierarchical structure of ViNG.

2.5.3 *Comparisons to Online Methods*

While Section 2.5.1 establishes that ViNG outperforms competitive offline methods for the task of goal-conditioned navigation, here we also investigate the performance of our method in comparison to popular online RL algorithms. Since the sample complexity of online RL algorithms forbids us from testing this in the real world, we use a Unity-based photorealistic outdoor navigation simulator. We include new additional baselines in the simulated experiment:

PPO: a popular reactive controller for indoor visual navigation algorithms [228, 283]

SoRB: online version of the "off-SoRB" baseline [65]

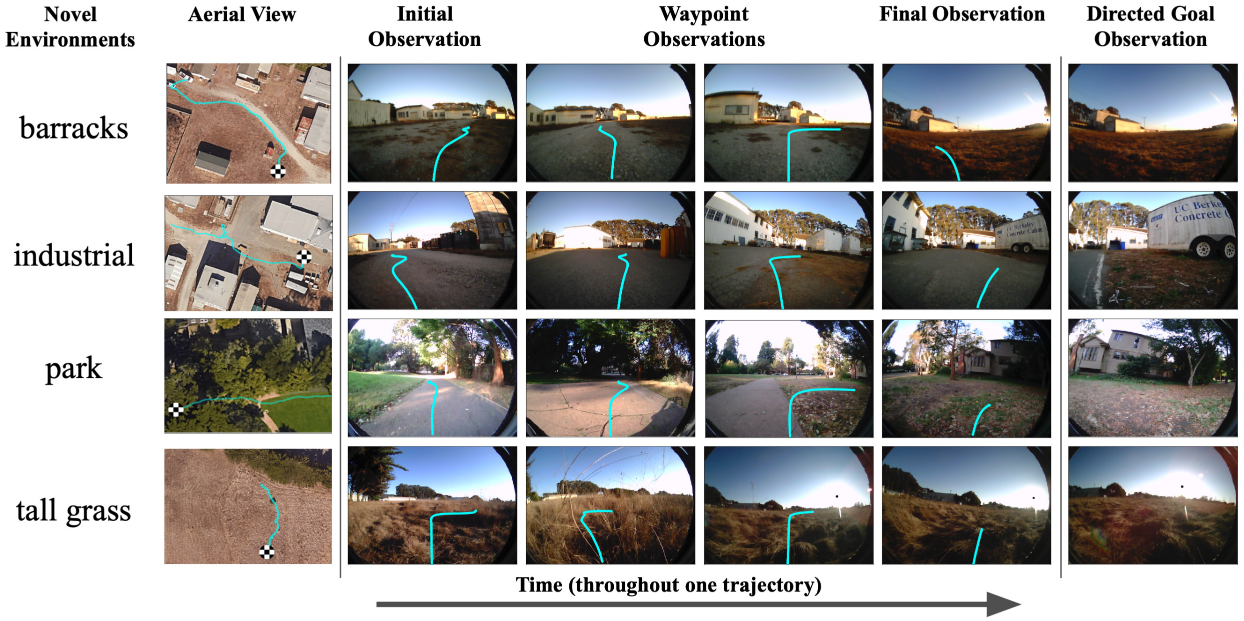


Figure 5: Generalization Experiments: We evaluate ViNG in four new outdoor environments. For each, we collect a few dozen minutes of experience to adapt the distance function and relative pose predictor. Then, given a goal image (last column, checkerboard location in aerial view), the robot attempts to navigate to the goal. Columns $4-7$ indicate that the robot succeeds in reaching the goal image. Cyan lines indicate the actions taken by ViNG.

We show results in Fig. 7. PPO performs poorly and is outperformed by ViNG, suggesting that a single image-based reactive policy is insufficient for solving long-horizon goal-reaching tasks, even when given access to 200 hours of online experience. SoRB outperforms other baselines and performs on par with ViNG. However, whereas ViNG requires 40 hours of offline data, SoRB requires 200 hours of online data, and must recollect this data for every experiment.

2.5.4 *Ablation Experiments*

A key design decision for ViNG that differentiates it from prior methods (e.g., [179, 65]) is how the controller generates actions to reach the next waypoint. We evaluate variants of ViNG that use alternative controllers and present results in Table 2. Two simple baselines, “direct actions” and “direct actions (discrete)”, use the goal-conditioned behavior cloning method of [224, 57] to directly predict (discrete) actions from the current and goal observations, without utilizing the topological map. Recall that our method uses the planner to command waypoints and then uses the relative pose together with a PD controller to reach each waypoint. We compared against a baseline that uses a different low-level controllers to reach these same waypoints: “Waypoint, Discrete” takes actions

Environment	ViNG Source	ViNG Target	ViNG Finetune	SPTM Finetune
barracks	0.27	0.42	**0.96**	0.74
industrial	0.13	0.44	**0.84**	0.68
park	0.04	0.32	**0.82**	0.71
tall grass	0	0.38	**0.79**	0.56

Table 1: Generalization Results: Our approach to generalization ("ViNG -Finetune") successfully navigates learns to navigate in four new environments (shown in Fig. 5) using just 60 minutes of experience in the new environment. Baselines that use only experience from the source or target domains are substantially less successful. Applying our finetuning approach on top of SPTM shows some generalization, but is outperformed by ViNG-Finetune.

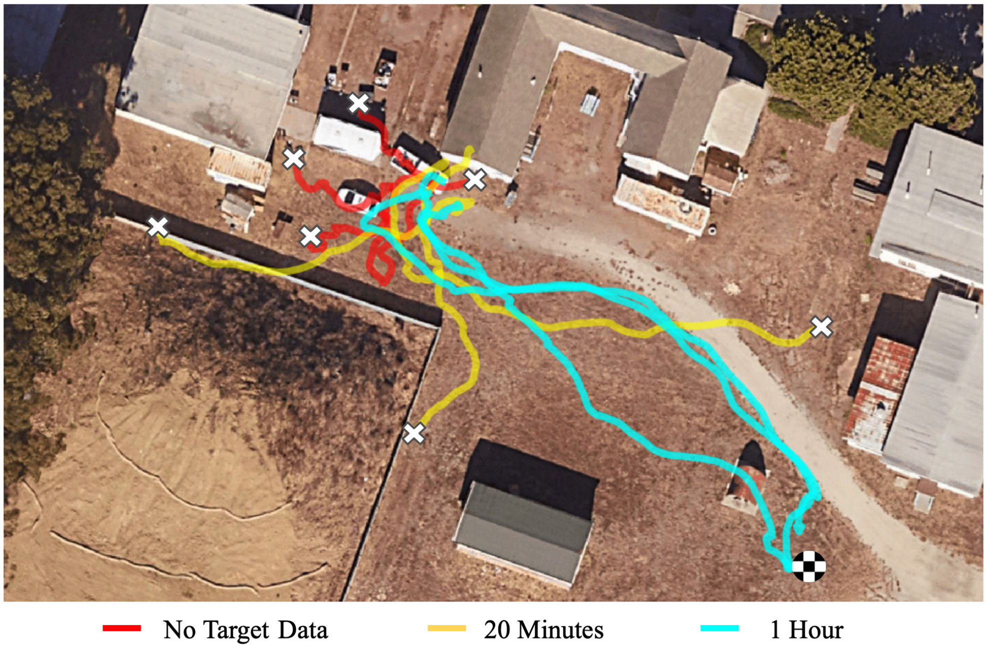


Figure 6: Fast Adaptation to a New Environment: After training ViNG in one environment, we deploy the system in a novel environment, shown above. By practicing to reach self-proposed goals and using that experience to finetune the controller, ViNG is able to quickly gain competence at reaching distance goals in this new environment, using just 60 minutes of experience. Example rollouts towards a goal 35m away (marked by checkerboard circle) demonstrate ViNG self-improving from interactions in the barracks environment.

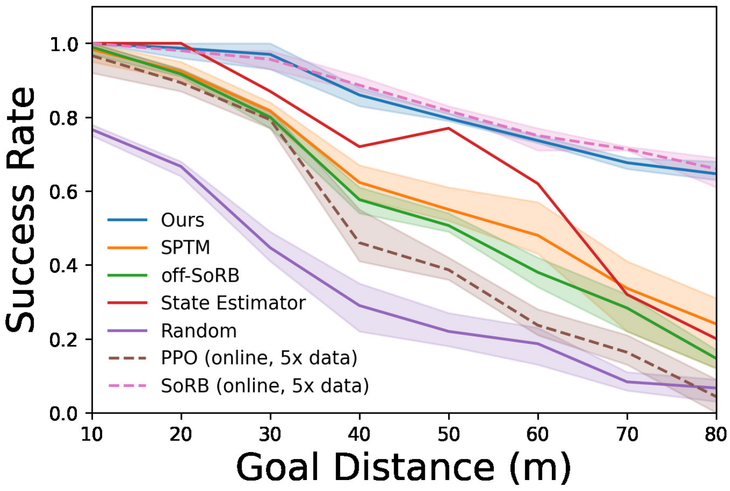


Figure 7: Results from Simulated Navigation: ViNG is substantially more successful at reaching distance goals than all offline baselines, while performing competitively with SoRB, a popular online baseline combining Q-learning and topological graphs. We emphasize that SoRB and PPO require 5× online data collection, making them prohibitively expensive to apply in the real-world.

using the "direction actions (discrete)" controller described above. As an alternative training scheme, "TD Waypoint" is a variant of our method that learns the traversability function via TD learning instead of supervised learning. Finally, we compare to two ablations of our method that skip the graph pruning and negative sampling stages of ViNG.

2.5.5 *Applications and Qualitative Results*

ViNG's ability to navigate using perception and landmarks, without access to maps or localization, can enable a number of intuitive applications, which we illustrate through qualitative results in this section. We constructed two demonstrations that reflect potential applications of our system:

1. *Contactless Last-Mile Delivery*: We demonstrate last-mile delivery in a residential complex by using ViNG to autonomously deliver mail and food to visually-indicated delivery locations. In this setting, users specify delivery destinations for the robot simply by taking a photograph of the desired destination, and the robot autonomously navigates to this destination to deliver a package.

2. *Autonomous Inspection*: Densely constructed building complexes, like university campuses, are often unmapped or lack accurate spatial localization. We reprogram ViNG to periodically navigate to landmarks, specified as images, around the campus to set up an autonomous patrolling system. Discrepancies can be identified by

	Success Rate @ Distance d (m)				
Controller	$d=10$	$d=20$	$d=30$	$d=40$	$d=50$
Direct Actions (Discrete)	0.87	0.81	0.74	0.65	0.45
Direct Actions	0.98	0.89	0.74	0.73	0.4
Waypoint, Discrete	**1.0**	0.95	0.91	0.82	0.7
Waypoint	**1.0**	**1.0**	**0.95**	0.88	0.81
TD Waypoint	**1.0**	**1.0**	**0.96**	**0.87**	**0.87**
Waypoint, No Pruning	**1.0**	**0.88**	0.81	0.79	0.52
Waypoint, Only Positives	**1.0**	0.91	0.75	0.76	0.43

Table 2: Ablation Experiments: We investigate design choices for the parametrization of the controller. Using waypoints as a mid-level action space is key to the performance of ViNG, which is particularly emphasized for distant goals. While training the models, we show that ViNG can be trained with either supervised or TD learning and report similar performance. We also show that the two key ideas presented – graph pruning and negative sampling – are indeed essential for the performance of ViNG in the real-world.

comparing the observations to previous observations (stored in the topological graph).

Figures 8 and 9 show ViNG successfully performing these tasks in the urban environment. Videos of the qualitative results, generalization experiments, and real-world applications can be found at the project website (sites.google.com/view/ving-robot).

2.6 DISCUSSION

In this paper, we proposed ViNG: a system for goal-directed navigation using visual observations and goals on an outdoor ground robot. While conceptually similar to prior methods, we demonstrate that a few key design choices, such as pruning the topological graph, parametrizing the controller in terms of a relative pose predictor and sampling negatives while training to minimize distribution shift, allow ViNG to learn to successfully navigate using only offline experience, a setting in which many prior methods fail. Intriguingly, we also demonstrate that ViNG can be quickly adapted to navigate in new environments. These generalization and self-improvement attributes highlight that learning-based approaches are not only an effective mechanism for handling high-dimensional observations, but are also amenable to fast adaptation to novel environments. Further, we have demonstrated ViNG on a number of real-world applications in dense, urban environments that may be unmapped or GPS-denied, and specifying visual goals is convenient – contactless last-mile delivery and autonomous inspection.

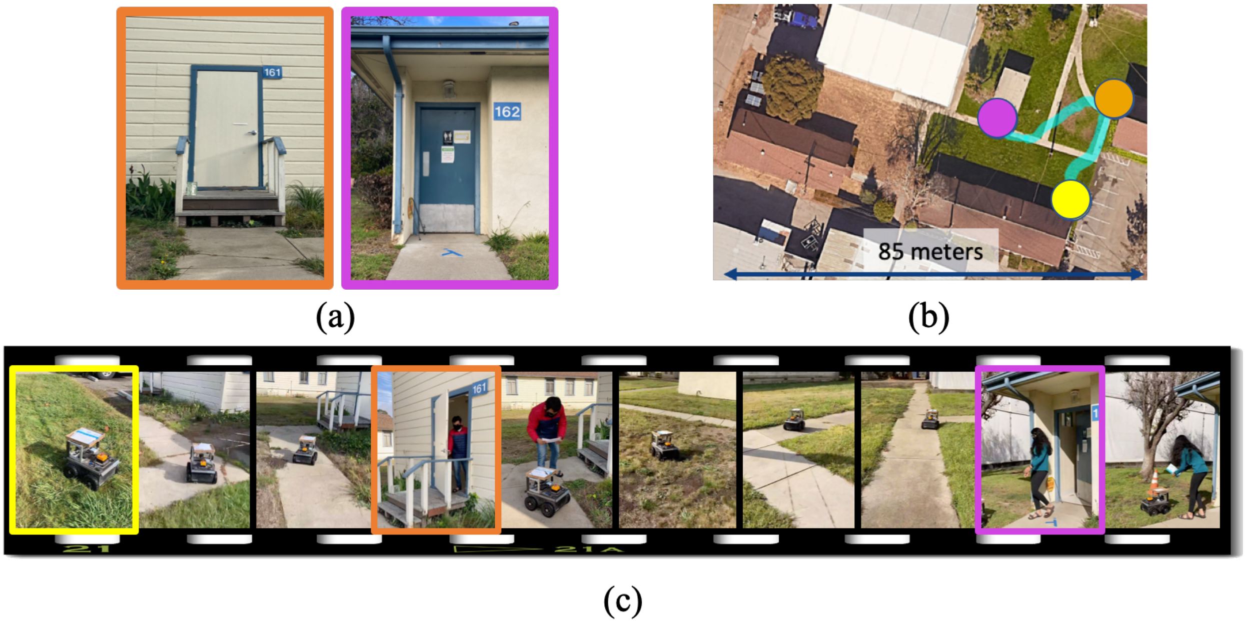


Figure 8: Contactless Last-Mile Delivery Demo: Given a set of visually-indicated goals (a), ViNG can perform contactless delivery in the urban neighborhood successfully, as shown in the filmstrip (c). An overhead view (b) with starting position marked in yellow and respective goals marked in orange and magenta shows the trajectory of the robot (cyan). *Note: The satellite view (b) is solely for visualization and is not available to the robot.*

Our method requires a static, offline dataset of observations over which we can plan. Many real-world tasks are non-stationary, with the distribution of observations shifting over time (e.g., lighting changes, dynamic objects, etc.). In future work, we aim to incorporate representations of observations and goals that are robust to such distributional shifts, which would expand the generalization capabilities of our method.

ACKNOWLEDGMENTS

This research was funded by the Office of Naval Research, DARPA Assured Autonomy, and ARL DCIST CRA W911NF-17-2-0181, with computing support from Google and Amazon Web Services. The authors would like to thank Jonathan Fink and Ethan Stump for their help setting up the simulation environment used for developing this research.

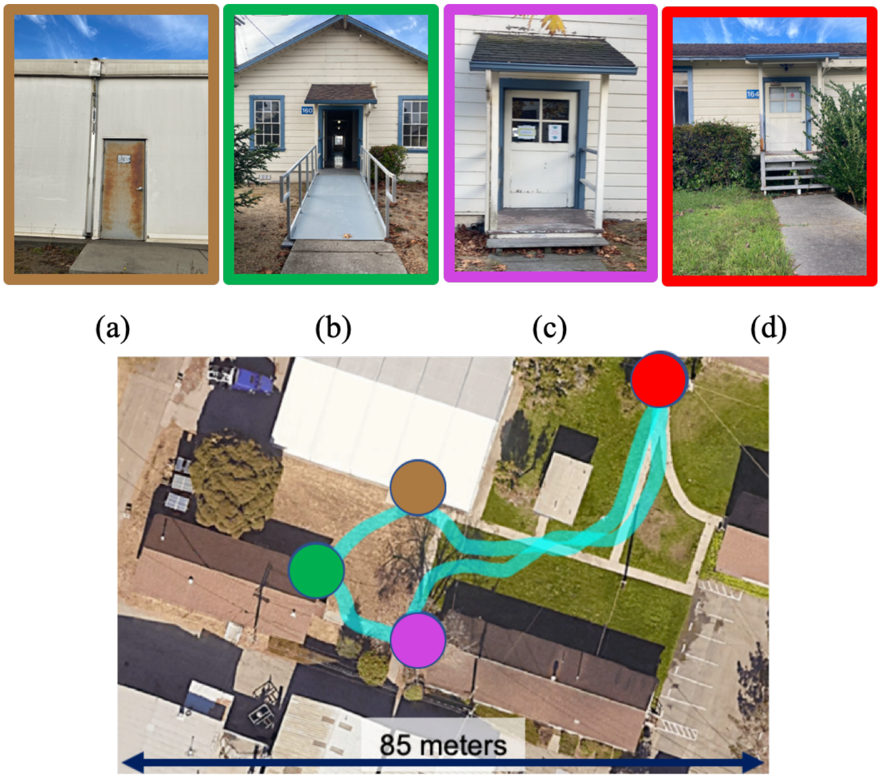


Figure 9: Autonomous Inspection Demo: Given a set of visual landmarks (a–d) in a university campus, ViNG can perform autonomous inspection by navigating to these goals periodically. An overhead view (b) shows color-coded goals and the trajectory taken by robot (cyan) in one cycle. *Note: The satellite view (e) is solely for visualization and is not available to the robot.*

3

OPEN-WORLD EXPLORATION WITH LATENT GOAL MODELS

Synopsis

In this chapter, we build upon our robotic learning system by enabling it to perform autonomous exploration and navigation in diverse, open-world environments. Our core idea is to use a learned latent variable model of distances and actions, along with a non-parametric topological memory of images. We use an information bottleneck to regularize the learned policy, giving us (i) a compact visual representation of goals, (ii) improved generalization capabilities, and (iii) a mechanism for sampling feasible goals for exploration. Trained on a large offline dataset of prior experience, the model acquires a representation of visual goals that is robust to task-irrelevant distractors. We demonstrate our method on a mobile ground robot in open-world exploration scenarios. Given an image of a goal that is up to 80 meters away, our method leverages its representation to explore and discover the goal in under 20 minutes, even amidst previously-unseen obstacles and weather conditions.

3.1 INTRODUCTION

Robustness is a key challenge in learning to navigate diverse, real-world environments. A robotic learning system must be robust to the difference between an offline training dataset and the real world (i.e., it must generalize), be robust to non-stationary changes in the real world (i.e., it must ignore visual distractors), and be equipped with mechanisms to actively explore to gather information about traversability. Different environments may exhibit similar physical structures, and these similarities can be used to accelerate exploration of *new* environments. Learning-based methods provide an appealing approach for learning a representation of this shared structure using prior experience.

In this work, we consider the problem of navigating to a user-specified goal in a previously unseen environment. The robot has access to a large and diverse dataset

Shah, Eysenbach, Kahn, Rhinehart, Levine, "Rapid Exploration for Open-World Navigation with Latent Goal Models", in Annual Conference on Robot Learning (CoRL) 2021
Project website: sites.google.com/view/recon-robot

of experience from *other* environments, which it can use to learn general navigational affordances. Our approach to this problem uses an information bottleneck architecture to learn a compact representation of goals. Learned from prior data, this latent goal model encodes prior knowledge about perception, navigational affordances, and short-horizon control. We use a non-parametric memory to incorporate experience from the new environment. Combined, these components enable our system to learn to navigate to goals in a new environment after only a few minutes of exploration.

The primary contribution of this work is a method for exploring novel environments to discover user-specified goals. Our method operates directly on a stream of image observations, without relying on structured sensors or geometric maps. An important part of our method is a compressed representation of goal images that simultaneously affords robustness while providing a simple mechanism for exploration. Such a representation allows us, for example, to specify a goal image at one time of day, and then navigate to that same place at a different time of day: despite variation in appearance, the latent goal representations must be sufficiently close that the model can produce the correct actions. Robustness of this kind is critical in real-world settings, where the appearance of landmarks can change significantly with times of day and seasons of the year.

We demonstrate our method, **R**apid **E**xploration **C**ontrollers for **O**utcome-driven **N**avigation (RECON), on a mobile ground robot and evaluate against 6 competitive baselines spanning over 100 hours of real-world experiments in 8 distinct open-world environments (Fig. 10). Our method can discover user-specified goals up to 80m away after just 20 minutes of interaction in a new environment. We also demonstrate robustness in the presence of visual distractors and novel obstacles. We make this dataset publicly available as a source of real-world interaction data for future resesarch.

3.2 RELATED WORK

Exploring a new environment is often framed as the problem of efficient mapping, posed in terms of information maximization to guide the robot to uncertain regions of the environment. Some prior exploration methods use local strategies for generating control actions for the robots [143, 20, 135, 260], while others use use global strategies based on the frontier method [291, 30, 101]. However, building high-fidelity geometric maps can be hard without reliable depth information. Further, such maps do not encode semantic aspects of traversability, e.g., tall grass is traversable but a wire fence is not.

Building on the system from Chapter 2, we construct a topological map by learning a distance function and a low-level policy. We estimate distances via supervised regression and learn a local control policy via goal-conditioned behavior cloning [82, 165]. However, these prior methods do not describe how to learn to navigate in *new*, unseen environments. We equip RECON with an explicit mechanism for exploring new environments and transferring knowledge across environments.

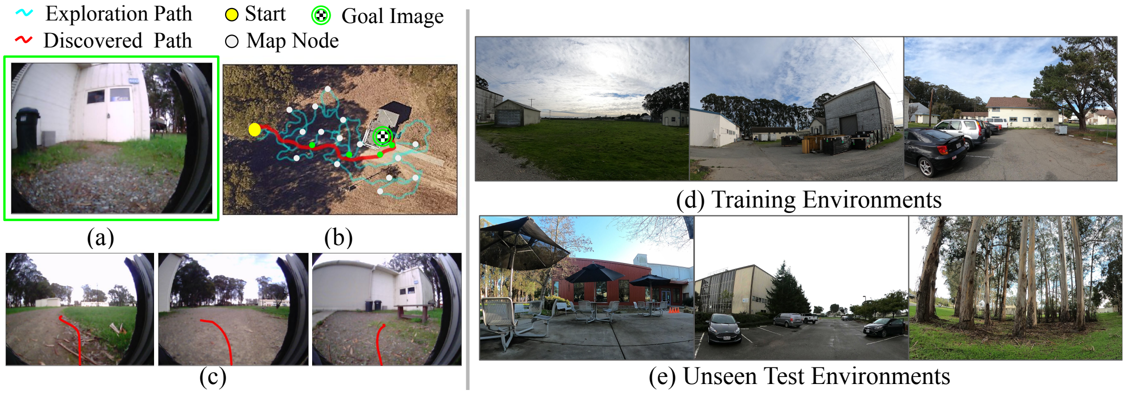


Figure 10: System overview: Given a goal image *(a)*, RECON explores the environment *(b)* by sampling prospective *latent* goals and constructing a topological map of images (white dots), operating only on visual observations. After finding the goal *(c)*, RECON can reuse the map to reach arbitrary goals in the environment (red path in *(b)*). RECON uses data collected from diverse training environments *(d)* to learn navigational priors that enable it to quickly explore and learn to reach visual goals a variety of unseen environments *(e)*.

Well-studied methods for exploration in reinforcement learning (RL) utilize a novelty-based bonus, computed from a predictive model [256, 198, 12, 24, 223, 226, 28], information gain [103, 181], or methods based on counts, densities, or distance from previously-visited states [14, 92, 152]. However, these methods learn to reason about the novelty of a state only after visiting it. Recent works [250, 29] improve upon this by predicting explorable areas for interesting parts of the environment to accelerate visual exploration. While these methods can yield state-of-the-art results in simulated domains [171, 136], they come at the cost of high sample complexity (over 1M samples) and are infeasible to train in open-world environments without a simulated counterpart. Instead, our method enables the robot to explore an environment from scratch in just 20 minutes, using prior experience from other environments.

The problem of reusing experience across tasks is studied in the context of meta-learning [62, 219, 184] and transfer learning [262, 150, 197, 88, 79]. Our method uses an information bottleneck [268], which serves a dual purpose: first, it provides a representation that can aid the generalization capabilities of RL algorithms [112, 84], and second, it serves as a measure of task-relevant uncertainty [6], allowing us to incorporate prior information for proposing goals that are functionally-relevant for learning control policies in the new environment.

The problem of learning goal-directed behavior has been studied extensively using RL [122, 227, 203, 67] and imitation learning (IL) [58, 82, 165, 257, 255, 213]. Our method builds upon prior goal-conditioned IL methods to solve a slightly different problem: how to reach goals in a *new* environment. Once placed in a new environment, our method explores by carefully choosing which goals to visit, inspired by prior work [204, 46, 300, 201, 161]. Unlike these prior methods, however, our method makes use of previous experience in *different* environments to accelerate learning in the current environment.

3.3 PROBLEM STATEMENT AND SYSTEM OVERVIEW

We consider the problem of goal-directed exploration for visual navigation in novel environments: a robot is tasked with navigating to a goal location G, given an image observation o_g taken at G. Broadly, this consists of three separate stages: (1) learning from offline data, (2) building a map in a new environment, and (3) navigating to goals in the new environment. We model the task of navigation as a Markov decision process with time-indexed states $s_t \in \mathcal{S}$ and actions $a_t \in \mathcal{A}$. We *do not assume* the robot has access to spatial localization or a map of the environment, or access to the system dynamics. We use videos of robot trajectories in a variety of environments to learn general navigational skills and build a compressed representation of the perceptual inputs, which can be used to guide the exploration of novel environments. We make no assumption on the nature of the trajectories: they may be obtained by human teleoperation, self-exploration, or as a result of a preset policy. These trajectories need not exhibit intelligent behavior. Since the robot only observes the world from a single on-board camera and does not run any state estimation, our system operates in a partially observed setting. Our system commands continuous linear and angular velocities.

3.3.1 *Mobile Robot Platform*

We implement RECON on a Clearpath Jackal UGV platform. The default sensor suite consists of a 6-DoF IMU, a GPS unit for approximate global position estimates, and wheel encoders to estimate local odometry. In addition, we added a forward-facing 170° field-of-view RGB camera and an RPLIDAR 2D laser scanner. Inside the Jackal is an NVIDIA Jetson TX2 computer. The GPS and laser scanner can become unreliable in some environments [124], so we use them solely as safety controllers during data collection. Our method operates only using images taken from the onboard RGB camera, without other sensors or ground-truth localization.

3.3.2 *Self-Supervised Data Collection & Labeling*

Our aim is to leverage data collected in a wide range of different environments to enable the robot to discover and learn to navigate to novel goals in novel environments. We curate a dataset of self-supervised trajectories collected by a time-correlated random walk in diverse real-world environments (see Fig. 10 (d,e)). This data was collected over a span of 18 months and exhibits significant variation in appearance due to seasonal and lighting changes. We make this dataset publicly available[1] and provide further details in Appendix A.1.

3.4 RECON : A METHOD FOR GOAL-DIRECTED EXPLORATION

Our objective is to design a robotic system that uses visual observations to efficiently discover and reliably reach a target image in a previously unseen environment. RECON consists of two components that enable it to explore new environments. The first is an uncertainty-aware, context-conditioned representation of goals that can quickly adapt to novel scenes. The second component is a topological map, where nodes represent egocentric observations and edges are the predicted distance between them, constructed incrementally from frontier-based exploration, maintaining a compact memory of the target environment.

3.4.1 *Learning to Represent Goals*

Our method learns a compact representation of goal images that is robust to task-irrelevant factors of variation. We learn this representation using a variant of the information bottleneck architecture [6, 1]. We use a context-conditioned representation of goals to learn a control policy in the target environment. Letting $I(\cdot;\cdot)$ denote mutual information, the objective in Eq. 1 encourages the model to compress the incoming goal image o_g into a representation z_t^g conditioned on the current observation o_t that is predictive of the best action a_t^g and the temporal distance d_t^g to the goal (upper-case denotes random variables):

$$I\big((A_t^g, D_t^g); Z_t^g \mid o_t\big) - \beta I(Z_t^g; O_g \mid o_t) \tag{1}$$

Following [6], we approximate the intractable objective in Eq. 1 with a variational posterior and decoder (an upper bound), resulting in the maximization objective:

$$L = \frac{1}{|\mathcal{D}|} \sum_{(o_t, o_g, a_t^g, d_t^g) \in \mathcal{D}} \mathbb{E}_{p_\phi(z_t^g | o_g, o_t)} \left[\log q_\theta\left(a_t^g, d_t^g \mid z_t^g, o_t\right)\right] - \beta \mathrm{KL}\left(p_\phi(\cdot \mid o_g, o_t) || r(\cdot)\right) \tag{2}$$

where we define the prior $r(z_t^g) \triangleq \mathcal{N}(0, I)$ and $\mathcal{D}$ is a dataset of trajectories characterized by (o_t, o_g, a_t^g, d_t^g) quadruples. The first term measures the model's ability to predict actions and distances from the encoded representation, and the second term measures the model's compression of incoming goal images.

As the encoder p_ϕ and decoder q_θ are conditioned on o_t, the representation z_t^g only encodes information about *relative* location of the goal from the context – this allows the model to represent *feasible* goals. If, instead, we had a typical VAE (in which the

[1] Available for download at sites.google.com/view/recon-robot/dataset.

Algorithm 3 *RECON for Exploration:* RECON takes as input an encoder p_ϕ, a decoder q_θ, prior r, the current observation o_t and goal observation o_g. $\delta_1, \delta_2, \epsilon, \beta \in \mathbb{R}_+; H, \gamma \in \mathbb{N}$ are hyperparameters.

```
 1: function RECON ($q_\theta, p_\phi, r, o_t, o_g; \delta_1, \delta_2, \epsilon, \beta, \gamma, H$)
 2:     $\mathcal{G} \leftarrow \emptyset, \mathcal{D} \leftarrow \emptyset$                ▷ Initialize graph and data
 3:     while not reached goal $[\bar{d}_t^g < \delta_1]$ do            ▷ Continue while not at goal
 4:         $o_n \leftarrow$ LeastExploredNeighbor($\mathcal{G}, o_t; \delta_2$)
 5:         $z_t^g \sim p_\phi(z \mid o_t, o_g)$                    ▷ Encode relative goal
 6:         if goal is feasible $[r(z_t^g) > \epsilon]$ then
 7:             $z_t^w \leftarrow z_t^g$                            ▷ Will go to the goal
 8:         else if robot at frontier $[\bar{d}_t^n < \delta_1]$ then
 9:             $z_t^w \sim r(z)$                             ▷ Will explore from frontier
10:         else
11:             $z_t^w \sim p_\phi(z \mid o_t, o_n)$                  ▷ Will go to frontier
12:         end if
13:         $\mathcal{D}_w, o_t \leftarrow$ SubgoalNavigate($z_t^w; H$)
14:         $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_w$
15:         ExpandGraph($\mathcal{G}, o_t$)
16:         Step $L(\phi, \theta; \mathcal{D}, \beta)$ for $\gamma$ epochs                 ▷ Eq. 2
17:     end while
18:     return networks $p_\phi, q_\theta$ and graph $\mathcal{G}$
19: end function
```

input images are autoencoded), the samples from the prior over these representations would not necessarily represent goals that are reachable from the current state. This distinction is crucial when exploring *new* environments, where most states from the training environments are not valid goals.

3.4.2 *Goal-Directed Exploration with Topological Memory*

The second component of our system is a topological memory constructed incrementally as the robot explores a new environment. It provides an estimate of the exploration frontier as well as a map that the robot can use to later navigate to arbitrary goals. To build this memory, the robot uses the model from the previous section to propose *subgoals* for data collection. Note that this is done in the exploration phase, where we have a latent goal model pre-trained on the offline dataset. Given a subgoal, our algorithm (Alg. 3) proceeds by executing actions towards the subgoal for a fixed number of timesteps (Alg. 3 L13). The data collected during subgoal navigation expands the topological memory (Alg. 3 L15) and is used to fine-tune the model (Alg. 3 L16). Thus, the task of efficient exploration is reduced to the task of choosing subgoals.

Algorithm 4 *RECON for Goal-Reaching:* After exploration, RECON uses the topological graph $\mathcal{G}$ to quickly navigate towards the goal o_g.

```
procedure GoalNavigate($\mathcal{G}, o_t, o_g; H$)
    $v_t \leftarrow \text{AssociateToVertex}(\mathcal{G}, o_t)$
    $v_g \leftarrow \text{AssociateToVertex}(\mathcal{G}, o_g)$
    $(v_t, \ldots, v_g) \leftarrow \text{ShortestPath}(\mathcal{G}, v_t, v_g)$
    for $v \in (v_t, \ldots, v_g)$ do
        $z \leftarrow p_\phi(z \mid o_t, o_g = v.\text{o})$
        $\mathcal{D}_w, o_t \leftarrow \text{SubgoalNavigate}(z; H)$
    end for
end procedure
```

Subgoals are represented by latent variables in our model, which may either come from the posterior $p_\phi(z|o_t, o_g)$, or from the prior $r(z)$. Given a subgoal z and observation o_t, the model decodes it into an action and distance pair $q(a_t^g, d_t^g|z, o_t)$; the action is used to control the robot towards the goal, and the distance is used to construct edges in the topological graph. The choice of intermediate subgoal to navigate toward at any step is based on the robot's estimate of the goal reachability and its proximity to the frontier. To determine the frontier of the graph, we track the number of times each node in the graph was selected as the navigation goal; nodes with low counts are considered to be on the frontier. In the following, we use $\bar{z}_t^g$ to denote the mean of the encoder $p_\phi(z \mid o_t, o_g)$, and $\bar{d}_t^g$ to denote the distance component of the mean of the decoder $q_\theta(a_t, d_t \mid \bar{z}_t^g, o_t)$ (i.e., the predicted number of time steps from o_t to $\bar{z}_t^g$). The choice of subgoal at each step is made as follows:

(i) Feasible Goal: The robot believes it can reach the goal and adopts the representation of the goal image as the subgoal (Alg. 3 L7). The robot's confidence in reaching the goal is based on the probability of the current goal embedding z_t^g under the prior $r(z)$. Large $r(z_t^g)$ implies the relationship between the observation o_t and the goal o_g is *in-distribution*, suggesting that the model's estimates of the distances is reliable – intuitively, this means that the model is confident about the distance to o_g and can reach it.

(ii) Explore at Frontier: The robot is at the "least-explored node" (frontier) o_n and explores by sampling a random conditional subgoal latent z_t^w from the prior (Alg. 3 L9). The robot determines whether it is at the frontier based on the distance (estimated by querying the model) to its "least explored neighbor" $\bar{d}_t^n$ – the node in the graph within a distance threshold (δ_2) of the current observation that has the lowest visitation count. If the distance to this node $\bar{d}_t^n$ is low (threshold δ_1), then the robot is at the frontier.

(iii) Go to Frontier: The robot adopts its "least-explored neighbor" o_n as a subgoal (Alg. 3 L11).

The SubgoalNavigate function rolls out the learned policy for a fixed time horizon H to navigate to the desired subgoal latent z_t^w, by querying the decoder $q_\theta(a_t, d_t|z_t^w, o_\tau)$ with a fixed subgoal latent. The endpoint of such a rollout is used to update the visitation counts in the graph $\mathcal{G}$. At the end of each trajectory, the ExpandGraph subroutine is used to update the edge and node sets $\{\mathcal{E}, \mathcal{V}\}$ of the graph $\mathcal{G}$ to update the representation of the environment. We provide the pseudocode for these subroutines in Appendix A.2.1. We

Method	Expl. Time (mm:ss) ↓	Nav. Time (mm:ss) ↓	SCT [294] ↑
PPO + RND [24]	21:18	00:47	0.22
InfoBot [84]	23:36	00:48	0.21
Active Neural SLAM (ANS) [28]	21:00	00:45	0.33
ViNG [241]	19:48	00:34	0.60
Ours + Episodic Curiosity (ECR) [226]	14:54	00:31	0.73
RECON (Ours)	**09:54**	**00:26**	**0.92**

Table 3: Exploration and goal reaching performance: Exploring 8 real-world environments, RECON reaches the goal 50% faster than the best baseline (ECR). ANS takes up to 2x longer to find the goal and NTS [250] fails to find the goal in every environment. On subsequent traversals, RECON navigates to the goal 20–85% faster than other baselines, and exhibits $> 30\%$ higher weighted success.

also share broader implementation details including choice of hyperparameters, model architectures and training details in Appendix A.2.2.

3.4.3 *System Summary*

RECON uses the latent goal model and topological graph to quickly explore new environments and discovers user-specified goals. Our complete system consists of three stages:

A) *Prior Experience:* The goal-conditioned distance and action model (Sec. 3.4.1) is trained using experience from previously visited environments. Supervision for training our model is obtained by using time steps as a proxy for distances and a relabeling scheme (Appendix A.1).

B) *Exploring a Novel Environment:* When placed in a new environment, RECON uses a combination of frontier-based exploration and latent goal-sampling with the learned model. The learned model is also fine-tuned to this environment. These steps are summarized in Alg. 3 and Sec. 3.4.2.

C) *Navigating an Explored Environment:* Given an explored environment (represented by a topological graph $\mathcal{G}$) and the model, RECON uses $\mathcal{G}$ to navigate to a goal image by planning a path of subgoals through the graph. This process is summarized in Alg. 4.

3.5 EXPERIMENTAL EVALUATION

We designed our experiments to answer four questions:

Q1. How does RECON compare to prior work for visual goal discovery in novel environments?

Q2. After exploration, can RECON leverage its experience to navigate to the goal efficiently?

Q3. What is the range of perturbations and non-stationary elements to which RECON is robust?

Q4. How important are the various components of RECON, such as sampling from an information bottleneck and non-parametric memory, to its performance?

3.5.1 Goal-Directed Exploration in Novel Environments

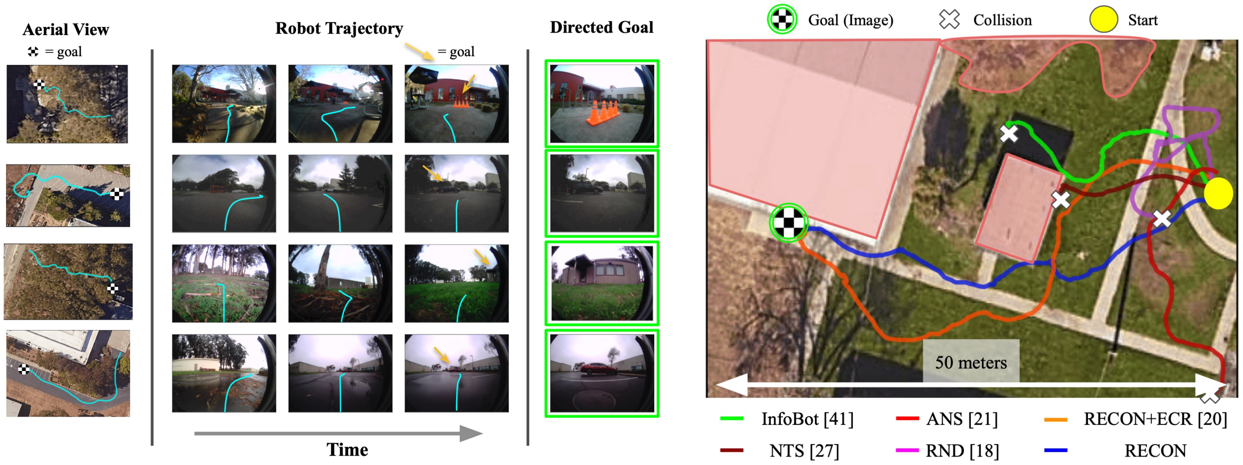


Figure 11: Visualizing goal-reaching behavior of the system: *(left)* Example trajectories to goals discovered by RECON in *previously unseen* environments. *(right)* Policies learned by the different methods in one such environment. Only RECON and ECR reach the goal successfully, and RECON takes the shorter route.

We perform our evaluation in a diverse variety of outdoor environments (examples in Fig. 10), including parking lots, suburban housing, sidewalks, and cafeterias. We train our self-supervised navigation model using an offline navigation dataset (Sec.3.3.2) collected in a distinct set of training environments, and evaluate our system's ability to discover user-specified goals in previously unseen environments. We compare RECON to five baselines, each trained on the same 20 hours of offline data as our method, and finetuned in the target environment with online interaction.

1. **PPO + RND:** Random Network Distillation (RND) is a widely used prediction bonus-based exploration strategy in RL [24], which we use with PPO [228, 283]. This

comparison is representative of a frequently used approach for exploration in RL using a novelty-based bonus.

2. **InfoBot:** An offline variant of InfoBot [84], which uses goal-conditioned information bottleneck, analogous to our method, but does not use the non-parametric memory.

3. **Active Neural SLAM (ANS):** A popular indoor navigation approach based on metric spatial maps proposed for coverage-maximizing exploration [28]. We adapt it to the goal-directed task by using the distance function from RECON to detect when the goal is nearby.

4. **Visual Navigation with Goals (ViNG):** A method that uses random action sequences to explore and incrementally build a topological graph without reasoning about visitation counts [241].

5. **Episodic Curiosity (ECR):** A method that executes random action sequences at the frontier of a topological graph for exploration [226]. We implement this as an *ablation* of our method that samples random action sequences, rather than rollouts to sampled goals (Alg. 3 Line 7).

We evaluate the ability of RECON to discover visually-indicated goals in 8 *unseen* environments and navigate to them repeatedly. For each trial, we provide an RGB image of the desired target (one per environment) to the robot and report the time taken by each method to *(i)* discover the desired goal (**Q1**), and *(ii)* reliably navigate to the discovered goal a second time using prior exploration (**Q2**). Additionally, we quantify navigation performance using Success weighed by Completion Time (SCT), a success metric that takes into account the agent's dynamics [294]. We show quantitative results in Table 3, and visualize sample trajectories of RECON and the baselines in Fig. 11.

RECON outperforms all the baselines, discovering goals that are up to 80m away in under 20 minutes, including instances where no other baseline can reach the goal successfully. RECON+ECR and ViNG discover the goal in only the easier environments, and take up to 80% more time to discover the goal in those environments. RND, InfoBot and ANS are able to discover goals that are up to 25m away but fails to discover more distant goals, likely because using reinforcement learning for fine-tuning is data-inefficient. We exclude reporting metrics on NTS, which fails to successfully explore any environment, likely due to overfitting to the offline trajectories. Indeed, the simulation experiments reported in each of these online algorithms require upwards of 1M timesteps to adapt to new environments [84, 28, 250]. We attribute RECON's success to the context-conditioned sampling strategy (described in Sec. 3.4.1), which proposes goals that can accelerate the exploration of new environments.

We then study RECON's ability to quickly reach goals after initial discovery. Table 3 shows that RECON variants are able to quickly recall a feasible path to the goal. These methods create a compact topological map from experience in the target environment, allowing them to quickly reach previously-seen states. The other baselines are unsuccessful at recalling previously seen goals for all but the simplest environments. Fig. 11 shows an aerial view of the paths recalled by various methods in one of the environments. Only the RECON variants are successfully able to navigate to the checkerboard goal; all other baselines result in collisions in the environment. Further, RECON discovers a shorter path to the goal and takes 30% less time to navigate to it than ECR ablation.

3.5.2 *Exploring Non-Stationary Environments*

Outdoor environments exhibit non-stationarity due to dynamic obstacles, such as automobiles and people, as well as changes in appearance due to seasons and time of day. Successful exploration and navigation in such environments requires learning a representation that is invariant to such distractors. This capability is of central interest when using a non-parametric memory: for the topological map to remain valid when such distractors are presented, we must ensure the invariance of the learned representation to such factors (**Q3**). To test the robustness of RECON to unseen obstacles and appearance changes, we first had RECON explore in a new "junkyard" to learn to reach a goal image containing a blue dumpster (Fig. 12-a). Then, without any more exploration, we evaluated the learned goal-reaching policy when presented with *previously unseen* obstacles (trash cans, traffic cones, and a car) and and weather conditions (sunny, overcast, and twilight). Fig. 12 shows trajectories taken by the robot as it successfully navigates to the goal in scenarios with varying obstacles and lighting conditions. These results suggest that the learned representations are invariant to visual distractors that do not affect robot's decisions to reach a goal (e.g., changes in lighting conditions do not affect the trajectory to goal, and hence, are discarded by the bottleneck).

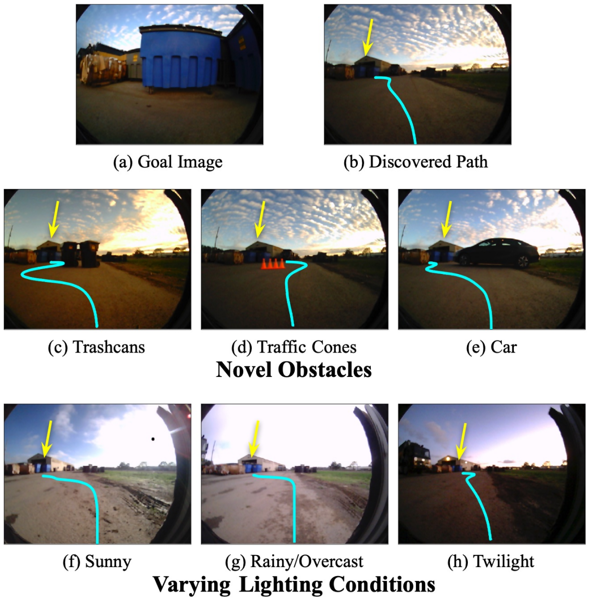


Figure 12: Exploring non-stationary environments: The learned representation and topological graph is robust to visual distractors, enabling reliable navigation to the goal under novel obstacles *(c–e)* and appearance changes *(f–h)*.

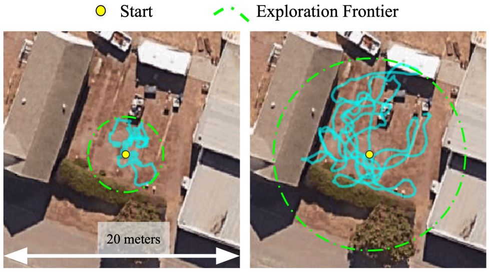


Figure 13: Exploration via sampling from our context-conditioned prior (*right*) allows the robot to explore 5 times faster than using random actions, e.g. in ECR [226] (*left*).

Method	Expl. Time ↓	Nav. Time ↓	SCT [294] ↑
Reactive	11:54	00:37.4	0.63
Rand. Actions	14:54	00:31.4	0.73
V. Sampling	14:06	00:28.7	0.83
Ours	**09:56**	**00:25.8**	**0.92**

Table 4: Ablation experiments confirm the importance of using an information bottleneck and a non-parametric memory.

3.5.3 *Dissecting RECON*

RECON explores by sampling goals from the prior distribution over state-goal representations. To quantify the importance of this exploration strategy (**Q4**), we deploy RECON to perform undirected exploration in a novel target environment *without building a graph* of the environment. We compare the coverage of trajectories of the robot over 5 minutes of exploration when: *(a)* it executes random action sequences [226], and *(b)* it performs rollouts towards sampled goals. We see that performing rollouts to sampled goals results in 5x faster exploration in novel environments (see Fig. 13).

We also evaluate several variants of RECON that ablate its goal sampling and non-parametric memory on the end-to-end task of visual goal discovery in novel environments:

- *Reactive:* our method deployed *without* the topological graph for memory.
- *Random Actions:* a variant of our method that executes random action sequences at the frontier rather than rollouts to sampled goals. This is identical to the ECR baseline described in Sec. 3.5.1.
- *Vanilla Sampling:* a variant of our method which learns a goal-conditioned policy and distances *without* an information bottleneck to obtain compressed representations.

We deploy these variants in a subset of the unseen test environments and summarize their performance in Table 4. These results corroborate the observations in Fig. 13: learning a compressed goal representation is key to the performance of RECON. "Vanilla Sampling", despite sampling from a joint prior, performs poorly and is unable to discover distant goals. We hypothesize that our method is more robust because the information bottleneck helps learn a representation that ignores task-irrelevant information. We also observe that "Reactive" experiences a smaller degradation in exploration performance, suggesting that goal-sampling can help with the exploration problem even without the graph. However, we find a massive degradation in its ability to recall previously discovered goals, suggesting that the memory is key to the navigation performance of RECON.

3.6 DISCUSSION

We proposed a system for efficiently learning goal-directed policies in new open-world environments. The key idea behind our method is to use a learned goal-conditioned distance model with a latent variable model representing visual goals for rapid goal-directed exploration. The problem setup studied in this paper, using past experience to accelerate learning in a *new* environment, is reflective of real-world robotics scenarios: collecting new experience at deployment time is costly, but experience from prior environments can provide useful guidance to solve new tasks.

In future work, we aim to provide theoretical guarantees for when and where we can expect stochastic policies and the information bottleneck to provide efficient exploration. One limitation of the current method is that it does not explicitly account for the value of information. Accounting for such states can generate a better goal-reaching policy.

ACKNOWLEDGMENTS

This research was supported by ARL DCIST CRA W911NF-17-2-0181, DARPA Assured Autonomy, and the Office of Naval Research. The authors would like to thank Suraj Nair and Brian Ichter for useful discussions, and Gregory Kahn for setting up the infrastructure used for autonomous collection of real-world data.

4

KILOMETER-SCALE EXPLORATION WITH GEOGRAPHIC HINTS

Synopsis

In this chapter, we extend our robotic learning system so it can utilize side information such as schematic roadmaps, satellite maps and GPS coordinates as a planning heuristic, to achieve kilometer-scale robot navigation and exploration in previously unseen environments. Our method, ViKiNG, incorporates a local traversability model, which looks at the robot's current camera observation and a potential subgoal to infer how easily that subgoal can be reached, as well as a heuristic model, which looks at overhead maps for hints and attempts to evaluate the appropriateness of these subgoals in order to reach the goal. These models are used by a heuristic planner to identify the best waypoint in order to reach the final destination. Our method performs no explicit geometric reconstruction, utilizing only a topological representation of the environment. Despite having never seen trajectories longer than 80 meters in its training dataset, ViKiNG can leverage its image-based learned controller and goal-directed heuristic to navigate to goals up to 3 kilometers away in previously unseen environments, and exhibit complex behaviors such as probing potential paths and backtracking when they are found to be non-viable. ViKiNG is also robust to unreliable maps and GPS, since the low-level controller ultimately makes decisions based on egocentric image observations, using maps only as planning heuristics.

4.1 INTRODUCTION

Robotic navigation has conventionally been approached as a geometric problem, where the robot constructs a 3D model of the environment and then plans a path through this model. End-to-end learning-based methods offer an alternative approach, where

Shah and Levine, "ViKiNG: Vision-Based Kilometer-Scale Navigation with Geographic Hints", Robotics: Science and Systems (RSS) 2022
Project website: sites.google.com/view/viking-release

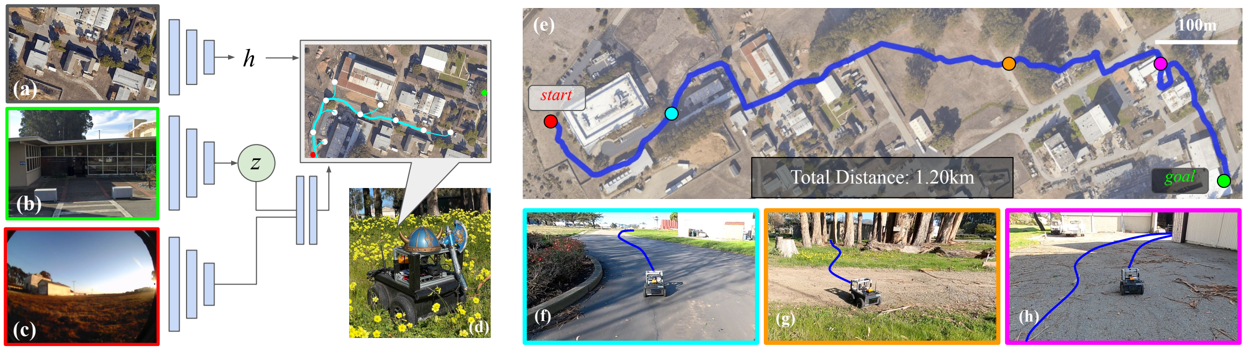


Figure 14: Kilometer-scale autonomous navigation with ViKiNG: Our learning-based navigation system takes as input the current egocentric image (c), a photograph of the desired destination (b), and an overhead map (which may be a schematic or satellite image) (a) that provides a *hint* about the surrounding layout. The robot (d) uses learned models trained in *other* environments to infer a path to the goal (e), combining local traversability estimates with global heuristics derived from the map. This enables ViKiNG to navigate *previously unseen* environments (e), where a single traversal might involve following roads (f), off-road driving under a canopy (g), and backtracking from dead ends (h).

the robot learns to correlate observations with traversability information directly from experience, without full geometric reconstruction [302, 34, 123]. This can be advantageous because, in many cases, geometry alone is neither necessary nor sufficient to traverse an environment, and a learning-based method can acquire patterns that are more directly indicative of traversability, for example by learning that tall grass is traversable [124] while seemingly traversable muddy soil should be avoided. More generally, such methods can learn about common patterns in their environment, such as that houses tend to be rectangular, or that fences tend to be straight. These patterns can lead to common-sense inferences about which path should be taken through an unknown environment even before that environment has been fully mapped out [188].

However, dispensing with geometry entirely may also be undesirable: the spatial organization of the world provides regularities that become important for a robot that needs to traverse large distances to reach its goal. In fact, when humans navigate new environments, they make use of both geographic knowledge, obtained from overhead maps or other cues, and learned patterns [282]. But in contrast to SLAM, humans don't require maps or auxiliary signals to be very accurate: a person can navigate a neighborhood using a schematic that roughly indicates streets and houses, and reach a house marked on it. Humans do not try to accurately reconstruct geometric maps, but use approximate "mental maps" that relate landmarks to each other topologically [74]. Our goal is to devise learning-enabled methods that similarly make use of geographic *hints*, which could take the form of GPS, roadmaps, or satellite imagery, without requiring these signals to be perfect.

We consider the problem of navigation from raw images in a *novel* environment, where the robot is tasked with reaching a user-designated goal, specified as an egocentric image, as shown in Figure 14. Note that the robot has *no prior experience* in the target environment.The robot has access to geographic side information in the form of a schematic roadmap or satellite imagery, which may be outdated, noisy, and unreliable, and approximate GPS. This information, while not sufficient for navigation by itself, contains useful cues that can be used by the robot. The robot also has access to a large and diverse dataset of experience from *other* environments, which it can use to learn general navigational affordances. We posit that an effective way to build such a robotic system is to combine the strengths of machine learning with informed search, by incorporating the geographic hints into a learned heuristic for search. The robot uses approximate GPS coordinates and an overhead map as geographic side information to help solve the navigation task, but does not assume that this information is particularly accurate—resembling a person using a paper map, the robot uses the GPS localization and an overhead map as *hints* to aid in visual navigation. Note that while we do assume access to GPS, the measurements are only accurate up to 2-5 meters (4-10× the scale of the robot), and cannot be used for local control.

The primary contribution of this work is ViKiNG, an algorithm that combines elements of end-to-end learning-based control at the low level with a higher-level heuristic planning method that uses this image-based controller in combination with the geographic hints. The local image-based controller is trained on large amounts of prior data from *other* environments, and reasons about navigational affordances directly from images without any explicit geometric reconstruction. The planner selects candidate waypoints in order to reach a faraway goal, incorporating the geographic side information as a planning heuristic. Thus, when the hints are accurate, they help the robot navigate toward the goal, and when they are inaccurate, the robot can still rely on its image observations to search through the environment. We demonstrate ViKiNG on a mobile ground robot and evaluate its performance in a variety of open-world environments not seen in the training data, including suburban areas, nature parks, and a university campus. Our local controller is trained on 42 hours of navigational data, and we test our complete system in 10 different environments. Despite never seeing trajectories longer than 80 meters in its training data, ViKiNG can effectively use geographic side information in the form of overhead maps to reach user-specified goals in *previously unseen environments* over 2 kilometers away in under 25 minutes.

4.2 RELATED WORK

Prior approaches have sought to incorporate learning into mapping and reconstruction [174, 116], which benefits from prior data, but aims at dense geometric reconstruction. Instead, approach uses a model that is trained with data from prior environments to

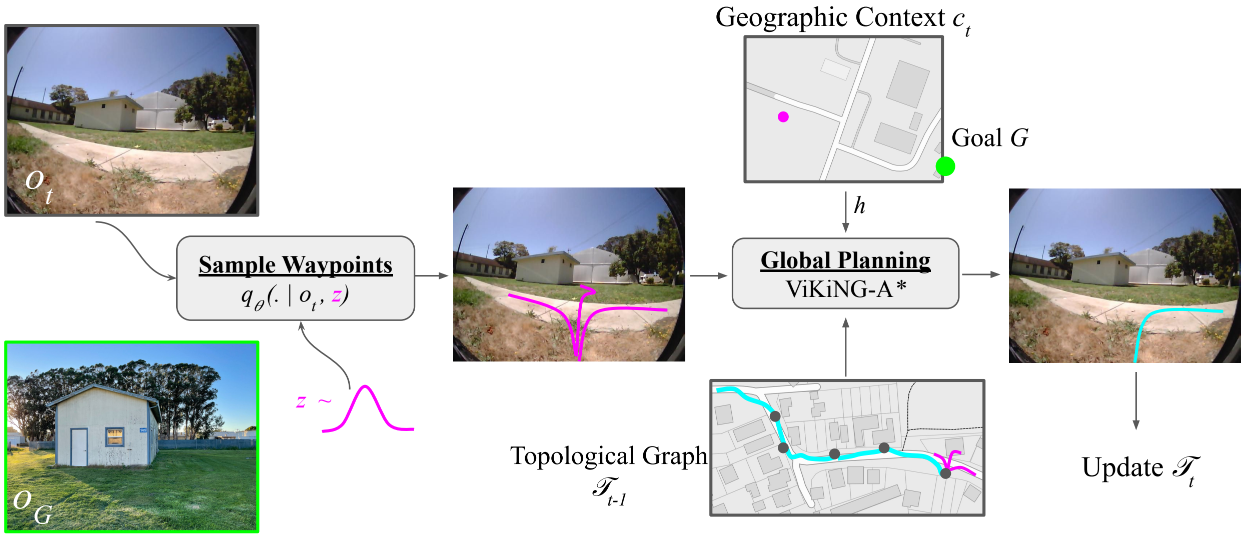


Figure 15: An overview of our method. ViKiNG uses latent subgoals z proposed by a learned low-level controller, which operates on raw image observations o_t, for global planning on a topological graph $\mathcal{T}$ to reach a distant goal o_G, indicates by a photograph and an approximate GPS location. A learned heuristic parses the overhead image c_t to bias this search towards the goal.

predict *traversability* rather than geometry, and this model is then used in combination with geographic hints to plan a path to the goal.

Prior works have studied the exploration problem by predicting explorable areas for semantically rich parts of the environment to accelerate visual exploration [250, 29]. While these methods can yield promising results in a variety of domains, they come at the cost of high sample complexity (over 10M samples) [171], making them difficult to use in the real world—the most performant algorithms take 10-20 minutes to find goals up to 50m away [238].

Following Chapter 3, our method trains a local model that predicts temporal distances and actions for nearby subgoals, and then incorporates this model into a search procedure that incrementally constructs a topological graph in a novel environment. Additionally, ViKiNG incorporates geographic hints in the form of approximate GPS coordinates and overhead maps. This enables ViKiNG to reach faraway goals, up to 25× further away than the furthest goal reported by RECON, and to reach goals up to 15× faster than RECON when exploring a novel environment.

4.3 VISUAL NAVIGATION WITH GEOGRAPHIC HINTS

Our aim is to design a robotic system that learns to use first-person visual observations to reach user-specified landmarks, while also utilizing geographic *hints* in the form of approximate GPS coordinates and overhead maps. At the core of our approach is a deep neural network that takes in the robot's current camera observation o_t, as well as an observation o_w of a potential subgoal w (we use "subgoal" and "waypoint" interchangeably), and predicts the time to reach w (or "temporal distance"), the best current action to do so, and the resulting spatial offset in terms of GPS coordinates. This model can also sample latent representations of potential *reachable* waypoints from the current observation o_t, which are used as candidate subgoals for planning. The model is trained on large amounts of data from a variety of training environments and, when the robot is placed in a *new* environment that it has not seen, it is used to incrementally construct a *topological* (non-geometric) graph to navigate to a distant user-specified goal. This goal is indicated by a photograph with an approximate GPS coordinate, and may be several kilometers away. The learned model alone is insufficient to navigate to such a distant goal in one shot, and therefore our planner uses a combination of the model's predictions and geographic information to plan a sequence of subgoals that search for a path through the environment, incrementally constructing the graph.

This process corresponds to a kind of heuristic search, where the geographic side information provides a heuristic to bias the robot to explore towards the goal as it constructs the topological graph. The latent goal model is used to determine reachability in this topological graph, and the geographic heuristic is used to steer the graph by exploring the environment. In a novel environment, the robot must incrementally build this graph using *physical search*, by visiting new nodes and expanding its frontier. The decision about *where* to actually go is determined by the first-person images, and the geographic information is used only as a heuristic, allowing ViKiNG to remain robust to noisy or unreliable side information. We overview our method in Figure 15.

4.3.1 *Low-level Control with a Latent Goal Model*

Our low-level model maps the current image observation o_t and a waypoint observation o_w to: (1) the temporal distance d_t^w to reach w from o_t; (2) the first action a_t^w that the robot must take now to reach w; (3) a prediction of the (approximate) offset in GPS readings between o_t and w, x_t^w. (1) and (3) will be used by the higher-level planner, and (2) will be used to drive to w, if needed. We would also like this model to be able to *propose*, in a learned latent space, potential subgoals w that are reachable from o_t, and predict their corresponding values of d_t^w, a_t^w, and x_t^w.

We present the model in Figure 16, with precise architecture details in the supplementary materials. The model is trained by sampling pairs of time steps in the trajectories in the training set. For each pair, the earlier time step image becomes o_t, and the later image becomes o_w. The number of time steps between them provides the supervision for d_t^w, the action taken at the earlier time step supervises a_t^w, and the later GPS reading is transformed into the coordinate frame of the earlier time step to provide supervision for x_t^w. The model is trained via maximum likelihood. Note that by training the model on

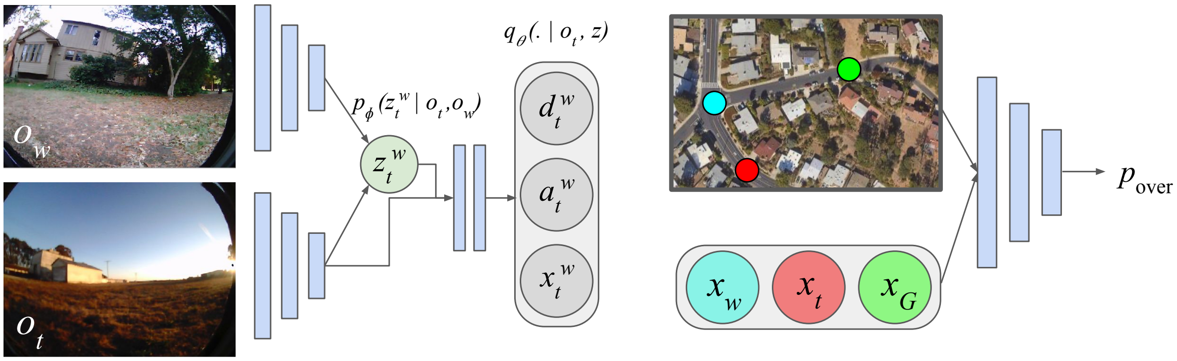


Figure 16: The learned models used by ViKiNG. The latent goal model (left) takes in the current image o_t. It also takes in either a true waypoint image o_w, or samples a *latent* waypoint $z_t^w \sim r(z_t^w)$ from a prior distribution, and then predicts, its temporal distance from o_t (d_t^w), the action to reach it (a_t^w), and its approximate GPS offset (x_t^w). The heuristic model (right) takes in an overhead image c_t, the approximate GPS coordinates of the current location (x_t) and destination (x_G), and the coordinates of the waypoint inferred by the latent goal model (x_w), and predicts an approximate heuristic value of the waypoint w for reaching the final destination.

data in this way, we not only enable it to evaluate reachability of prospective waypoints, but also make it possible to inherit behaviors observed in the data. For example, in our experiments, we will show that the model has a tendency to follow sidewalks and forest trails, a behavior it inherits from the portion of the dataset that is collected via teleoperation.

Besides predicting d_t^w, a_t^w, and x_t^w, our planner requires this model to be able to *sample* potential reachabale waypoints from o_t (see Figure 15). We implement this via a variational information bottleneck (VIB) inside of the model that bottlenecks information from o_w. Thus, the model can either take as input a real image o_w of a prospective waypoint, or it can sample a *latent* waypoint $z_t^w \sim r(z_t^w)$ from a prior distribution. We train the model so that sampled latent waypoints correspond to feasible locations that the robot can reach from o_t without collision.

Training the latent goal model: The full model, illustrated in Figure 16, can be split into three parts: a waypoint encoder $p_\phi(z_t^w|o_w, o_t)$, a waypoint prior $r(z_t^w)$, and a predictor $q_\theta(\{a, d, x\}_t^w|z_t^w, o_t)$. The latent waypoint representation z_t^w can either be sampled from the prior (which is fixed to $r(z_t^w) \triangleq \mathcal{N}(0, I)$), or from the encoder $p_\phi(z_t^w|o_w, o_t)$ if a waypoint image o_w is provided. This latent waypoint is used together with o_t to predict all desired quantities according to $q_\theta(\{a, d, x\}_t^w|z_t^w, o_t)$. The training set consists of tuples $(o_t, o_w, \{a, d, x\}_t^w)$, but the model must be trained so that samples $z_t^w \sim r(z_t^w)$ also produce valid predictions. We accomplish this by means of the VIB [6], which regularizes the encoder $p_\phi(z_t^w|o_w, o_t)$ to produce distributions that are close to the prior $r(z_t^w)$ in terms of KL-divergence. We refer the reader to prior work for a derivation of the VIB [6], and present our training objective for p_ϕ and q_θ below:

$$\begin{aligned}\mathcal{L}_{\text{VIB}}(\theta, \phi) = E_{\mathcal{D}}\big[-\mathbb{E}_{p_\phi}\left[\log q_\theta(\{a, d, x\}_t^w \mid z_t^w, o_t)\right] \\ + \beta \text{KL}\left(p_\phi(z_t^w \mid o_w, o_t) || r(z_t^w)\right)\big]\end{aligned} \tag{3}$$

The outer expectation over all tuples $(o_t, o_w, \{a, d, x\}_t^w) \in \mathcal{D}$ in the training distribution is estimating using the training set. The first term causes the model to accurately predict the desired information, while the second term forces the encoder to remain consistent with the prior, which makes the model suitable for sampling latent waypoints according to $z_t^w \sim r(z_t^w)$. As the encoder p_ϕ and decoder q_θ are conditioned on o_t, the representation z_t^w only encodes *relative* information about the subgoal from the context—this allows the model to represent *feasible* subgoals in new environments, and provides a compact representation that abstracts away irrelevant information, such as time of day or visual appearance. An analogous representation has been proposed in prior work [238], but did not predict spatial offsets and was used only for *uninformed* exploration without geographic hints.

4.3.2 *Informed Search on a Topological Graph*

The model described above can effectively reach nearby subgoals, for example those on which the robot has line of sight, but we wish to reach goals that are more than a kilometer away. To reach distant goals, we combine the model with a search procedure that incorporates geographic hints from satellite images or roadmaps. The system does not require this information to be accurate, instead using it as a planning heuristic while still relying on egocentric camera images for control. Our high-level planner plans over a topological graph $\mathcal{T}$ that it constructs incrementally using the low-level model in Section 4.3.1 as a local planner. We first describe a generic version of the algorithm for any heuristic, and then describe the data-driven heuristic function that we extract from the geographic hints via contrastive learning.

Challenges with *physical search*: Our "search" process involves the robot physically searching through the environment, and is not purely a computational process. In contrast to standard search algorithms (e.g., Dijkstra, A*, IDA*, D*, etc.), each "step" of our search involves the robot driving to a subgoal and updating the graph. Standard graph search algorithms assume (i) the ability to *visit* any arbitrary node, and (ii) access to a set of *neighbors* for every node and the corresponding "edge weight," before visiting each neighbor. Physical search with a robot violates these assumptions, since robots cannot "teleport" and *visiting* a node incurs a driving cost. Furthermore, the real world does not provide "edge weights" and the robot needs to estimate the cost to reach an unvisited node *before* actually driving to it.

An algorithm for informed physical search: To solve these challenges, we design ViKiNG-A*, an A*-like search algorithm that uses our latent goal model and a learned heuristic

Algorithm 5 *ViKiNG-A* for Physical Search*

```
1:  function ViKiNG-A*(start $S$, goal info $o_G$, $x_G$)
2:      $\Omega \leftarrow \{S\}$
3:      while $\Omega$ not empty do
4:          $w_t \leftarrow \min(\Omega, f)$
5:          DriveTo$(w_t)$                          ▷ update visitations $v$ on the way
6:          observe image $o_t$
7:          add $w_t$ to graph $\mathcal{T}$                  ▷ use $q_{\theta,\phi}$ on $o_t$ to get distances
8:          if close$(o_t, o_G)$ finish           ▷ use $q_{\theta,\phi}(\{a,d,x\}_t^w | o_t, o_G)$
9:          remove $w_t$ from $\Omega$
10:         sample waypoints $w$ near $w_t$ (Section 4.3.1)
11:         for each $w$ sampled near $w_t$ do
12:             if not contains$(\Omega, w)$ then add $w$
13:         end for
14:         for each waypoint $w \in \Omega$ do
15:             $f(w) = g(t,w) + d_{\mathrm{Pr}[w]}^w + h(w) + v(\mathrm{Pr}[w])$
16:         end for
17:     end while
18:     return failure
19: end function
```

to perform *physical search* in real-world environments. While ViKiNG-A* does prefer shorter paths, it does not aim to be optimal (in contrast to A*), only to reach the goal successfully. We will use a heuristic $h(w)$, fully described in the next section, which we assume provides a *comparative* evaluation of candidate waypoints in terms of their anticipated temporal distance to the destination. Algorithm 5 outlines ViKiNG-A*.

Like A*, ViKiNG-A* maintains a priority queue "open set" Ω of unexplored fringe nodes and a "current" node that represents the least-cost node in this set, which we refer to as w_t. It also maintains a graph with visited waypoints, $\mathcal{T}$, where nodes correspond to images seen at those nodes, and edges correspond to temporal distances estimated by the model in Section 4.3.1. At every iteration, the robot *drives* to the least-cost node in the open set (L5), using a procedure that we outline later. When it reaches w_t, it observes the image o_t using its camera (L6). This allows it to add o_t to the graph $\mathcal{T}$ (L7), connecting it to other nodes by evaluating the distances using the model in Section 4.3.1. The graph construction is analogous to prior work [241, 238]. If the robot is close to the final goal image o_G according to the model (L8), the search ends. o_t also allows it to sample nearby candidate waypoints using the model in Section 4.3.1 (L10): first sampling $z_t^w \sim r(z_t^w)$ from the prior, and then decoding distances d_t^w, a_t^w, and x_t^w, from which it can compute absolute locations as $x_w = x_t + x_t^w$. Each sampled waypoint is stored in the open set, and annotated with the current image o_t and d_t^w. We refer to w_t as the *parent* of w, and index it as $\Pr[w]$. Note that we do *not* have access to the image o_w, as we have not visited the sampled waypoint w yet, and therefore we must store the current image o_t instead. This also means that we *cannot* connect these waypoints to the graph $\mathcal{T}$ except through their parent. Next, we re-estimate the cost of each waypoint in the open set, including the newly added waypoints.

The cost for each waypoint $w \in \Omega$ from the current point w_t consists of four terms (L15): (1) $g(t, w)$, the cost to navigate to the *parent* of w, which is part of the graph $\mathcal{T}$; this can be computed as a shortest path on the graph $\mathcal{T}$, and is zero for the current node. (2) $d^w_{\Pr[w]}$, the distance from the parent of w to w itself. (3) $h(w)$, the heuristic cost estimate of reaching the final goal from w (see Section 4.3.3). (4) $v(\Pr[w])$, the visitation count of $\Pr[w]$, computed as $CN(\Pr[w])$, where C is a constant and $N(\Pr[w])$ is a count of how many times the robot drove to $\Pr[w]$ via the DriveTo subroutine; this acts as a *novelty bonus* to encourage the robot to explore novel states, a strategy widely used in RL [147, 14]. Summing these terms expresses a preferences for nodes that are fast to reach from w_t (1 + 2), get us closer to the goal (3), and have not been heavily explored before (4). At the next iteration (L4), the robot picks the lowest-cost waypoint and again drives to it.

To navigate to a selected waypoint w (DriveTo), the robot employs a procedure analogous to prior work on learning-based navigation with topological graphs [241, 238], planning the shortest path through $\mathcal{T}$, and selecting the next waypoint on this path. Once the waypoint w is selected, the model $q_{\theta,\phi}(\{a, d, x\}^w_t | o_t, o_w)$ is used to repeatedly choose the action a^w_t based on the current image o_t, until the distance d^w_t becomes small, indicating that the waypoint is reached and the robot can navigate to the next waypoint (in practice, it's convenient to replan the path at this point, as is standard in MPC). Each time the DriveTo subroutine reaches a node, it also increments its count $N(w)$ which is used for the novelty bonus $v(w)$. The helper function close uses the model in Section 4.3.1 to check if the estimated temporal distance d^w_t is less than ϵ for two observations, and the contains operation on a set checks if a given node is *close* to any node inside the set. These modifications allow A*-like operations on the nodes of our graph, which are continuous variables.

4.3.3 *Learning a Goal-Directed Heuristic for Search*

We now describe how we extract a heuristic h from geographic side information. As a warmup, first consider the case where we only have the GPS coordinates for a waypoint (x_w) and final goal (x_G). We can use $\|x_g - x_w\|$ as a heuristic to bias the search to waypoints in the direction of the goal, and this heuristic can be readily obtained from the model in Section 4.3.1. However, we would like to compute the heuristic function using some side information c_t, such as a roadmap or satellite image, that does not lie in a metric space. Thus, we need to *learn* the heuristic function from data. Since ViKiNG-A* does not aim to be optimal (only seeking a feasible path), we do not require the heuristic to be admissible.

We train the heuristic $h_{\text{over}}(x_w, x_G, x_t, c_t)$ to score the favorability of a sampled candidate waypoint w for reaching the goal G from current location x_t, given side information c_t. In our case, c_t is an overhead image that is roughly centered at the current location of the robot. Our heuristic is based on an estimator for the probability $p_{\text{over}}(w \to G | x_w, x_G, x_t, c_t)$ that a given waypoint w lies on a valid path to the goal G. We use the same training set as in Section 4.3.1 to learn a predictor for p_{over}. Given p_{over}, we can generate a heuristic $h_{\text{over}} := \lambda_{\text{over}}(1 - p_{\text{over}})$ to steer ViKiNG-A* towards the goal (Alg. 5 L14). Note that, since we evaluate the heuristic for sampled candidate waypoints, we do not have access to x_w, but we can predict it by using the model in Section 4.3.1 to infer the offset x_t^w using o_t and the sampled latent code, and then calculate x_w from x_t and x_t^w. Thus, the heuristic is technically a function of c_t, o_t, x_t, and x_G.

Our procedure for training $p_{\text{over}}(w \to G | x_w, x_G, x_t, c_t)$ is based on InfoNCE [194], a contrastive learning objective that can be seen as a binary classification problem between a set of *positives* and *negatives*. At each training iteration, we sample a random batch $\mathcal{B}$ of sub-trajectories k from our training set, where x_S is the start of k and x_E is the end, and c_S is an overhead image centered at x_S. We sample a positive example by picking a random time step in this subtrajectory, and using its position x_{w^+}. The negatives x_{w^-} are locations of other randomly sampled time steps from other trajectories, comprising the set $\mathcal{W}^-$. In this way, we train a neural network model to represent $p_{\text{over}}(w \to G | x_w, x_G, x_t, c_t)$ (see Figure 16, right) via the InfoNCE objective:

$$\mathcal{L}_{\text{NCE}} = -\mathbb{E}_{\mathcal{B}}\left[\log \frac{p_{\text{over}}(w^+ \to E | x_{w^+}, x_E, x_S, c_S)}{\sum_{w^- \in \mathcal{W}^-} p_{\text{over}}(w^- \to E | x_{w^-}, x_E, x_S, c_S)}\right] \tag{4}$$

This heuristic can only reason about waypoints and goals at the scale of individual trajectories in the training set (up to 50m). For kilometer-scale navigation, the heuristic needs to make predictions for goals that are much further away, so we take inspiration from goal chaining in reinforcement learning [32] and combine overlapping trajectories in the training set (according to GPS positions) into larger trajectory groups. For a batch $\mathcal{B}$ of trajectories, we combine two trajectories if they intersect in 2D space. The resulting macro-trajectories thus have multiple start and goal positions, and can extend for several kilometers. We then sample the sub-trajectories for x_S, x_E, and x_{w^+} from these much longer macro-trajectories, giving us positive examples between very distant x_S, x_E pairs. This allows p_{over} to be trained on a vast pool of long-horizon goals and improves the reliability of the heuristic. We provide more details about this procedure in Appendix B.1.

4.4 VIKING IN THE REAL WORLD

We now describe our experiments deploying ViKiNG in a variety of real-world outdoor environments for kilometer-scale navigation. Our experiments compare ViKiNG to other learning-based methods, evaluate its performance at different ranges, and study how it responds to degraded or erroneous geographic information.

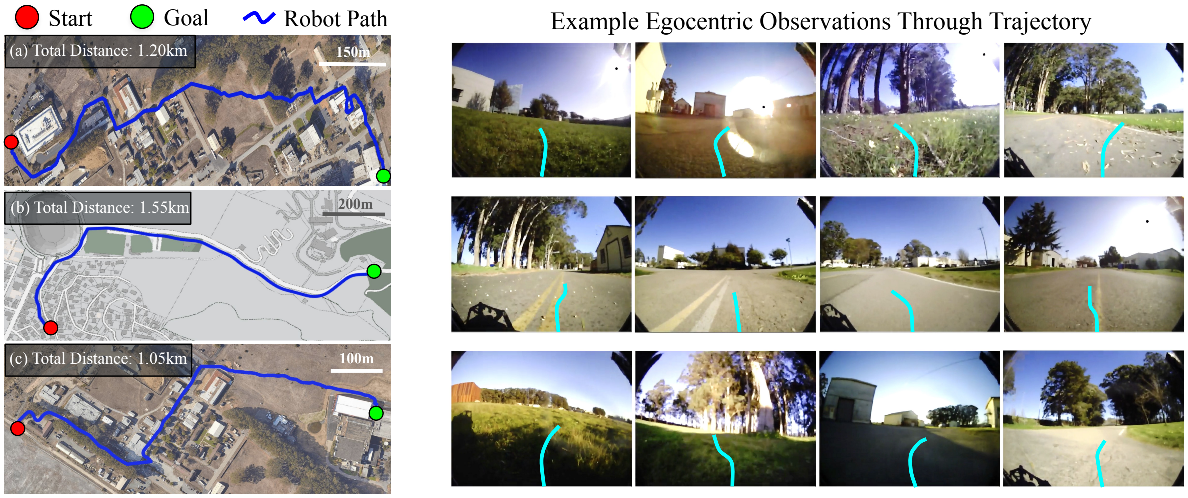

Start
Goal
Robot Path
(a) Total Distance: 1.20km
150m
(b) Total Distance: 1.55km
200m
(c) Total Distance: 1.05km
100m
Example Egocentric Observations Through Trajectory

Figure 17: Examples of kilometer-scale goal-seeking in *previously unseen* environments using only egocentric images (right) and a schematic roadmap or satellite image as *hints* (left). ViKiNG can navigate in complex environments composed of roads, meadows, trees and buildings.

4.4.1 *Mobile Robot Platform*

We implement ViKiNG on a Clearpath Jackal UGV platform (see Fig. 14). The default sensor suite consists of a 6-DoF IMU, a GPS unit for approximate global position estimates, and wheel encoders to estimate local odometry. Under open skies, the GPS unit is accurate up to 2-5 meters, which is 4-10× the size of the robot. In addition, we added a forward-facing 170° field-of-view RGB camera. Compute is provided by an NVIDIA Jetson TX2 computer, and a cellular hotspot connection provides for monitoring and (if necessary) teleoperation. Our method uses only the monocular RGB images from the onboard camera, unfiltered measurements from onboard GPS, and overhead images (roadmap or satellite) queried at the current GPS location, without any other processing.

4.4.2 *Offline Training Dataset*

Our aim is to leverage data collected in a wide range of different environments to (i) enable the robot to learn navigational affordances that generalize to novel environments, and (ii) learn a global planning heuristic to steer physical search in novel environments. To create a diverse dataset capturing a wide range of navigation behavior, we use 30 hours of publicly available robot navigation data collected using an autonomous, randomized data collection procedure in office park style environments [238]. We augmented this dataset with another 12 hours of teleoperated data collected by driving on city sidewalks, hiking trails, and parks. Notably, ViKiNG never sees trajectories longer than 80 meters, but is able to leverage the learned heuristic (Section 4.3.3) to reach goals over a kilometer away at over 80% of the average speed in the training set. The average trajectory length in the dataset is 45m, whereas our experiments evaluate runs in excess of 1km. The average velocity in the dataset is 1.68 m/s, and the average velocity the robot maintains in testing is 1.36 m/s. We provide more details about the dataset in Appendix B.2.

4.4.3 *Kilometer-Scale Testing*

For evaluation, we deploy ViKiNG in a variety of *previously unseen* open-world environments to demonstrate kilometer-scale navigation. Figure 17 shows the path taken by the robot in search for a user-specified goal image and location. ViKiNG is able to utilize geographic hints, in the form of a roadmap or satellite image centered at its current position, to steer its search of the goal. In a university campus (Fig. 17(a, c)), we observe that the robot can identify large buildings along the way and plan around it, rather than following a greedy strategy. Since the training data often contains examples of the robot driving around buildings, ViKiNG is able to leverage this prior experience and generalize to novel buildings and environments. On city roads (Fig. 17(b)), the learned heuristic shows preference towards following the sidewalks, a characteristic of the training data in city environments. It is important to note that while the robot has seen some prior data on sidewalks and in suburban neighborhoods, it has never seen the specific areas (see Appendix B.2 for further details). For videos of our experiments, please check out our project page.

These long-range experiments also exhibit successful backtracking behavior—when guided into a cul-de-sac by the planner, ViKiNG turns around and resumes its search for the goal from another node in the "openSet", reaching the goal successfully (see Figure 14(h)). While the learned heuristic provides high-level guidance, the local control is done solely from first person images. This is illustrated in Figure 14(g), where the robot navigates through a forest, where the satellite image does not contain any useful information about navigating under a dense canopy. ViKiNG is able to successfully navigate through a patch of trees using the image-based model described in Section 4.3.1. We can also provide ViKiNG with a set of goals to execute in a sequence to provide more guidance about the path (e.g., an inspection task with landmarks), as demonstrated in the next experiment.

A hiking ViKiNG: We deploy ViKiNG, with access to satellite images as hints, on a 2.7km hiking trail with a 70m elevation gain by providing a sequence of six checkpoint images and their corresponding GPS coordinates. Algorithmically, we run ViKiNG-A* on every goal (one at a time) while reusing the topological graph $\mathcal{T}$ across goals. Figure 18 shows a top-down view of the path taken by the robot—ViKiNG is able to successfully combine the strengths of a learned controller for collision-free navigation with a learned heuristic that utilizes the satellite images to encourage on-trail navigation between checkpoints. Since the offline dataset contains examples of trail-following, the robot learns to stay on trails when possible. This behavior is emergent from the data—there is no other

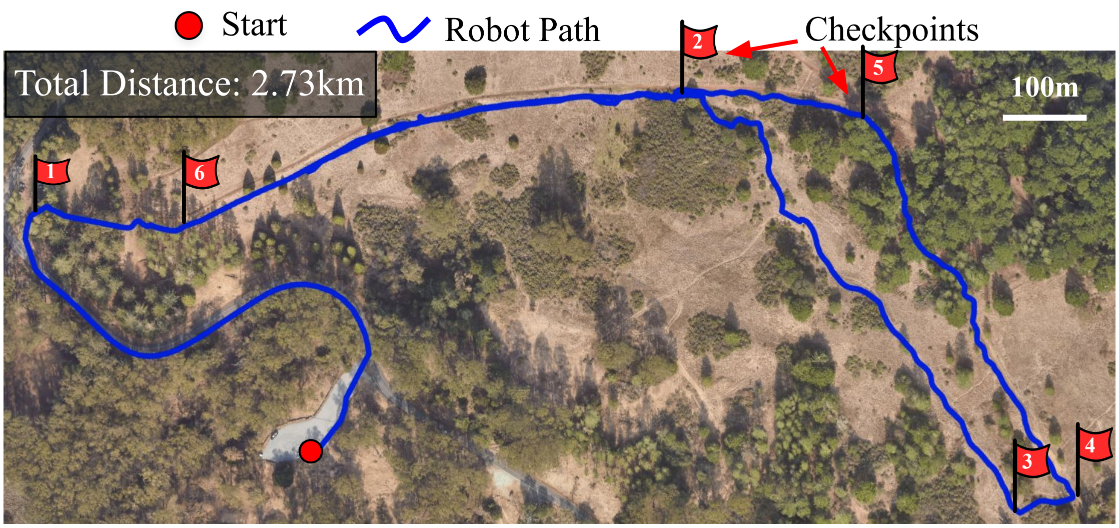


Figure 18: ViKiNG can follow a sequence of goal checkpoints to perform search in complex environments, such as this 2.73km hiking trail.

mechanism that encourages staying on the trails, and in several cases, a straight-line path between the goal waypoints would not stay on the trail (e.g., the first checkpoint in Figure 18).

Autonomous visual inspection: We further deploy ViKiNG in a suburban environment for the task of visual inspection specified by five images of interest. ViKiNG is able to successfully navigate to the landmarks by using satellite imagery, traveling a distance of 2.65km without any interventions. Figure 19 shows the specified images and a top-down view of the path taken by the robot on the trail.

4.4.4 *Quantitative Evaluation and Comparisons*

We compare ViKiNG to four prior approaches, each trained using the same offline data as our method. All methods have access to the egocentric images, GPS location, and satellite images, and control the robot via the same action space, corresponding to linear and angular velocities.

Behavioral Cloning: A goal-conditioned behavioral cloning (BC) policy trained on the offline dataset that maps the three inputs to control actions [44, 241].

PPO: A policy gradient algorithm that maps the three inputs to control actions. This comparison is representative of state-of-the-art "PointGoal" navigation in simulation [283].

GCG: A model-based algorithm that uses a predictive model to plan a sequence of actions that reach the goal without causing collision [123]. We use GCG in the goal-directed mode with a GPS target, using the onboard camera and satellite images as input modalities.

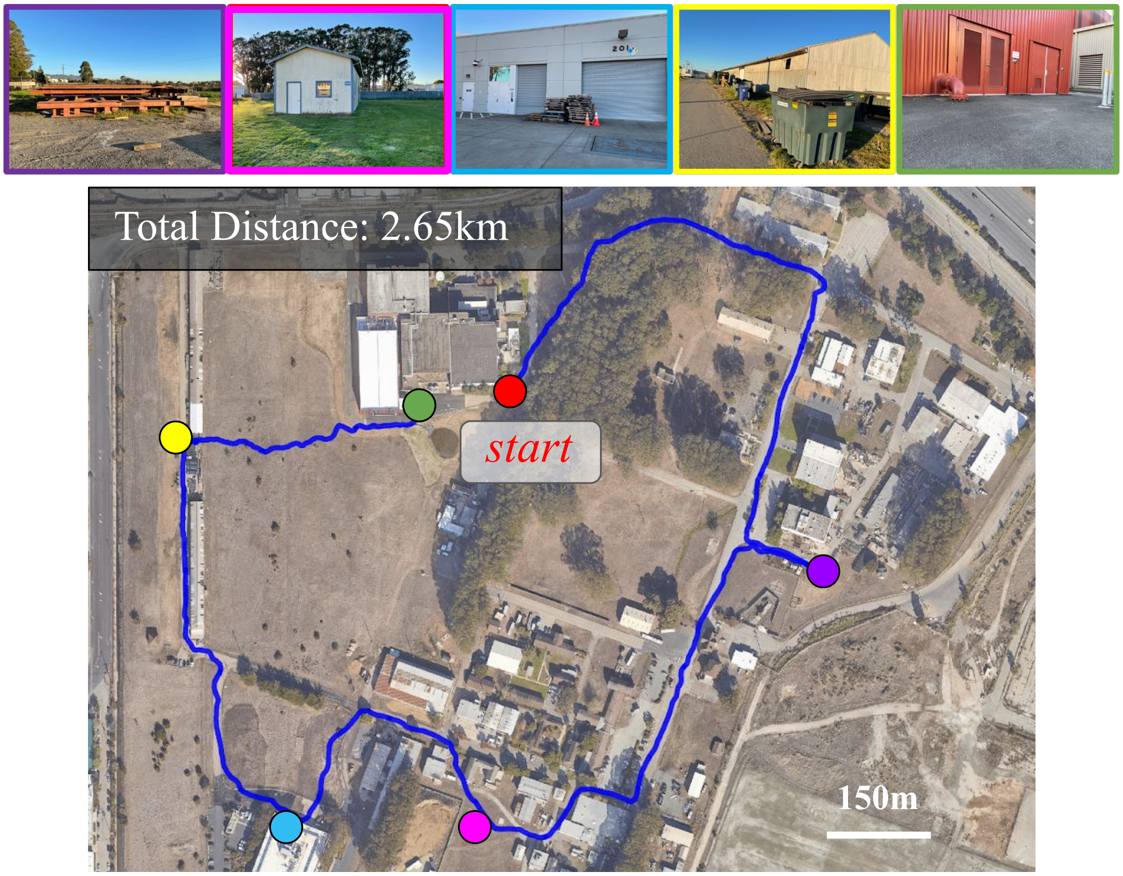


Figure 19: ViKiNG can utilize a satellite image to follow a sequence of visual landmarks (top) in complex suburban environments, such as this 2.65km loop stretching across buildings, meadows and roads.

RECON-H: A variant of RECON, which uses a latent goal model to represent reachable goals and plans over sampled subgoals to explore a novel environment [238]. We modify the algorithm to additionally accept the GPS and satellite images as additional inputs alongside the onboard camera image.

We evaluate the ability of ViKiNG to discover visually-indicated goals in 10 *unseen* environments of varying complexity. For each trial, we provide an RGB image of the desired target and its rough GPS location (accurate up to 5 meters). A trial is marked successful if the robot reaches the goal without requiring a human disengagement (due to a collision or getting stuck). We report the success rates of all methods in these environments in Table 5 and visualize overhead plots of the trajectories in one such environment in Figure 20.

ViKiNG outperforms all the prior methods, successfully navigating to goals that are over up to 500 meters away in our comparisons, including instances where no other method succeeds. RECON-H is the most performant of the other methods, successfully

Method	Easy < 50m	Medium $50 - 150$m	Hard $150 - 500$m
Behavior Cloning	2/3	1/4	0/3
Offline PPO [228]	2/3	1/4	0/3
GCG [123]	**3/3**	2/4	0/3
RECON-H [238]	**3/3**	3/4	1/3
ViKiNG (Ours)	**3/3**	**4/4**	**3/3**

Table 5: Comparison of goal-seeking performance against baselines. ViKiNG successfully reaches all goals. RECON-H and GCG succeed in simpler cases but are unable to utilize the hints effectively for distant goals. PPO and BC fail in all but the simplest cases.

Method	Avg. Displacement (m)	Avg. Velocity (m/s)
Behavior Cloning	19.5	0.35
Offline PPO [228]	47.2	0.85
GCG [123]	78.3	**1.40**
RECON-H [238]	188.3	0.41
ViKiNG (Ours)	**250.0+**	**1.36**

Table 6: Average robot displacement and velocity before disengagement. ViKiNG successfully reaches all goals without requiring any disengagements. RECON-H also reaches some distant goals, but the low avg. velocity suggests that it takes an efficient path.

reaching most goals in the easier environments. Visualizing the robot trajectories (Fig. 20) reveals that RECON-H is unable to successfully utilize the geographic hints and explores greedily on encountering an obstacle. It also gets stuck and is unable to backtrack in 2/10 instances. While GCG also performs well in simpler environments, it is limited by its planning horizon (up to 5 seconds) and gets stuck. PPO and BC both are both unable to learn from prior data and produce collisions with bushes and a parked car, respectively. In contrast, ViKiNG is able to effectively use the local controller to avoid the obstacles and reach the goal.

Analyzing the performance in the harder tasks with ranges of up to 500 meters (Table 6), the average displacements and velocities before a user disengagement (due to collision or getting stuck) during these runs further confirm that ViKiNG is able to effectively use the geographic hints to steer the search without running into obstacles. While RECON-H manages to reach some faraway goals, it takes a greedy path to do so and is over 3× slower than ViKiNG (see Fig. 20).

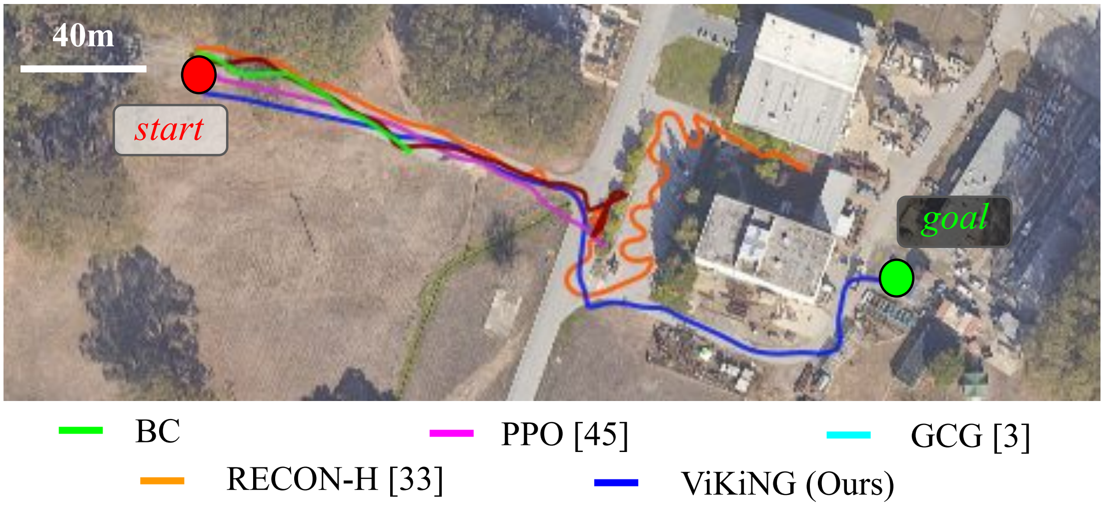


Figure 20: Trajectories taken by the methods in a *previously unseen* environment. Only ViKiNG is able to effectively use the overhead images to reach the goal (270m away) successfully, following a smooth path around the building. RECON-H and GCG get stuck, while PPO and BC result in collisions.

4.5 THE ROLE OF GEOGRAPHIC HINTS

In this section, we closely examine the role of geographic hints on the performance of ViKiNG by studying how it deals with a low-fidelity roadmap (versus a satellite image), and with incorrect hints and degraded geographic information. For the experiment in Section 4.5.1, we use models trained on the same dataset, but using schematic roadmaps as geographic hints. In Sections 4.5.2 and 4.5.3, we use the same satellite image model from Section IV, with no additional retraining to accommodate missing or imperfect geographic information.

4.5.1 *Comparing Different Types of Hints*

To understand the nature of hints learned by the heuristic for different sources of geographic side information, we compare two separate versions of ViKiNG: one trained with schematic roadmaps as hints, and another trained with satellite images. Note that the method is identical in both cases, only the hint image in the data changes. For identical start-goal pairs, we observe that a model trained with roadmaps prefers following marked roads, whereas one trained with satellite images often cuts across patches of traversable terrain (e.g., grass meadows or trails) to take the quicker path, despite being trained on the same data. We hypothesize that this is due to the ability of the learned models to extract better correlations from the feature-rich satellite images, in contrast to the more

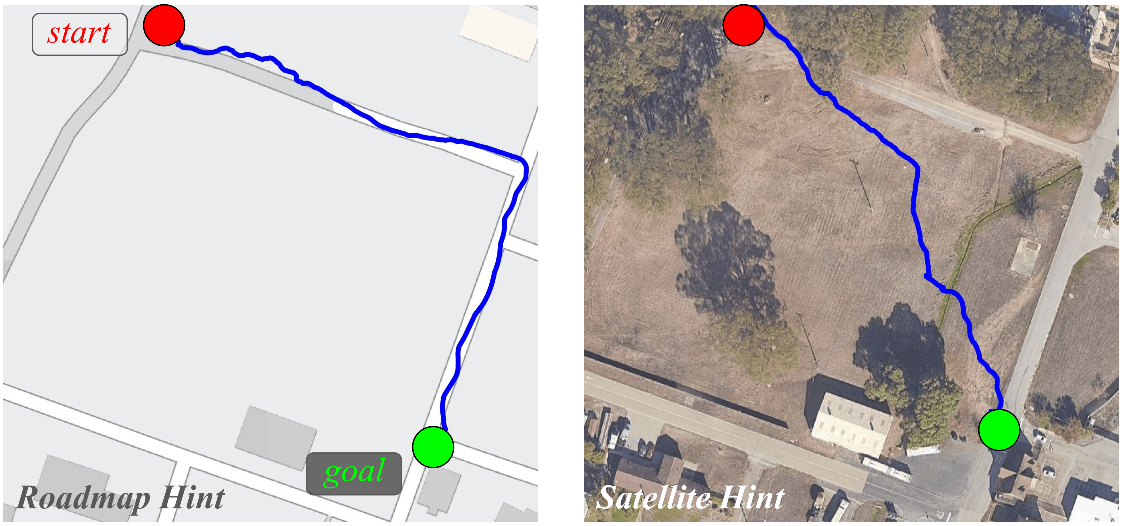


Figure 21: ViKiNG can use geographic hints in the form of a schematic roadmap or a satellite image. Providing roadmap hints encourages ViKiNG to follow marked roads (left); with satellite images, it is able to find a more direct path by cutting across a meadow (right).

abstract roadmap. Figure 21 shows a top-down view of the paths taken by the robot in the two cases in one such experiment.

4.5.2 *Outdated Hints*

To test the robustness of ViKiNG to outdated hints, we set up a goal-seeking experiment in one of the earlier environments and added a new obstacle—a large truck—blocking the path that ViKiNG took in the original trial. Since the satellite images are queried from a pre-recorded dataset, they do not reflect the addition of the truck, and hence continue to show a feasible path. We observe that the robot drives up to the truck and takes an alternate path to the goal, without colliding with it (see Figure 22). The lower-level latent goal model is robust to such obstacles and only proposes *valid* subgoal candidates that do not lead to collision; since the learned heuristic only evaluates valid subgoals, ViKiNG is robust to small discrepancies in the hints.

4.5.3 *Incorrect Hints*

Next, we set up a goal-seeking experiment in one of the easy environments with modified GPS measurements, so that the satellite images available to ViKiNG are offset by a ∼5km constant. As a result, this *hints* to the robot that there may be a road that it should follow, where in fact there isn't one (see Figure 23). We observe that the robot indeed deviates from its earlier path (with a valid map, the robot drives straight to the goal);

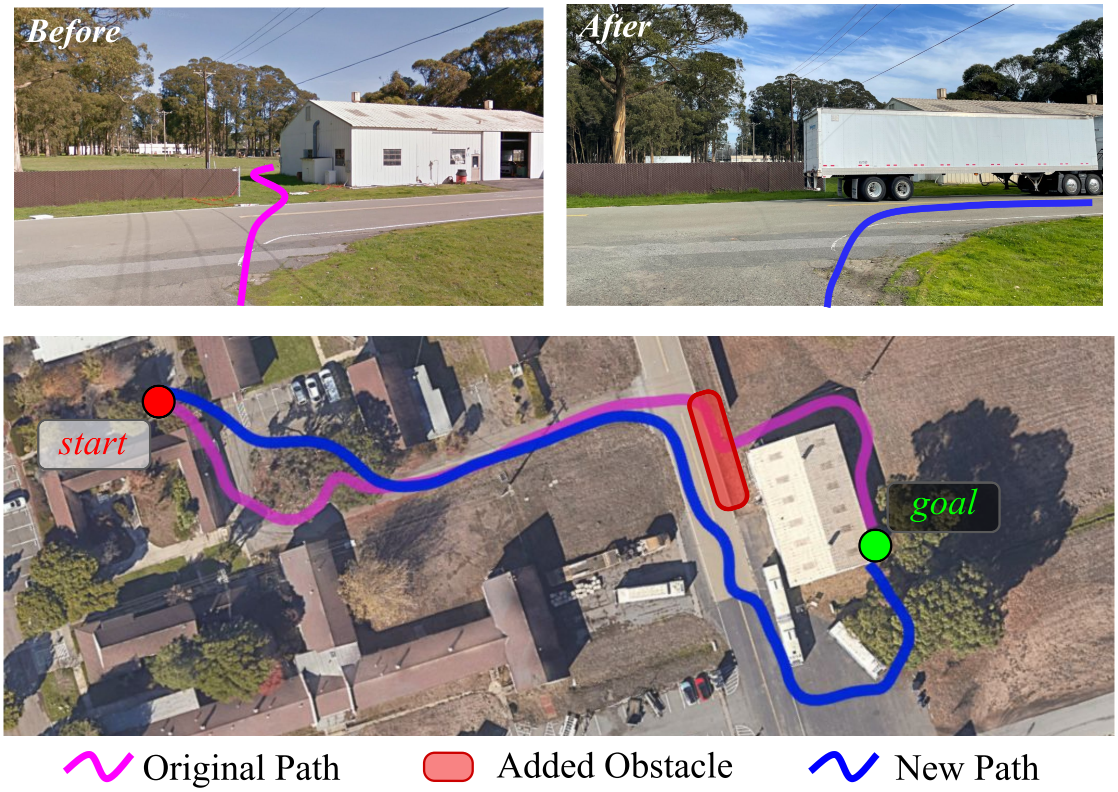


Figure 22: On navigating with outdated hints, like the truck (top right) that is absent in the satellite image, ViKiNG uses its learned local controller to propose feasible subgoals that avoid obstacles and finds a new path (blue) to the goal that avoids the truck.

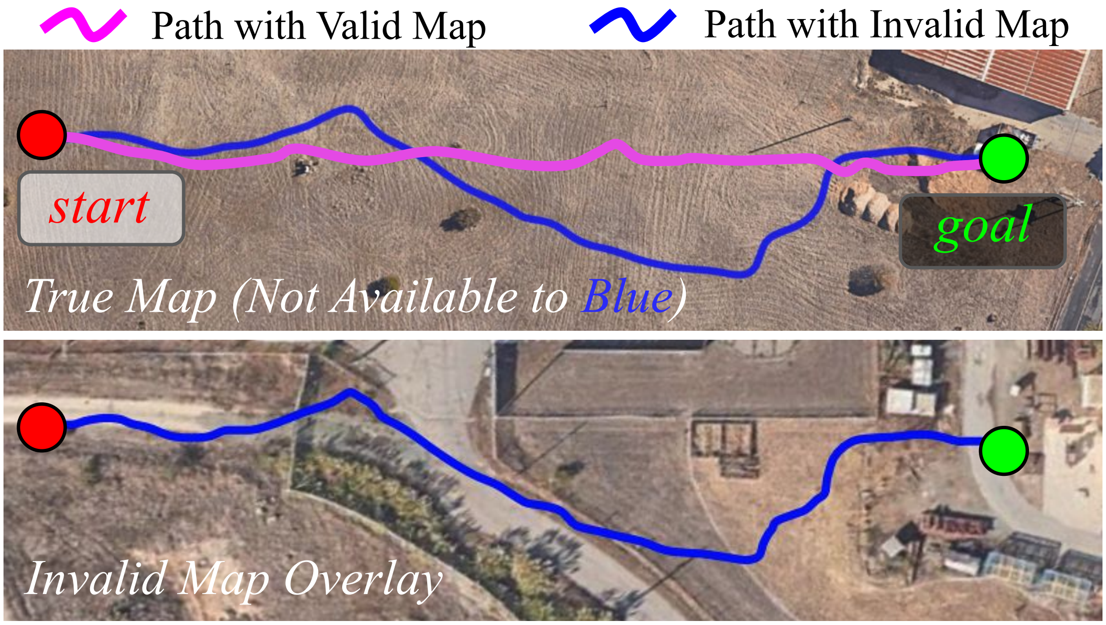


Figure 23: On navigation with invalid hints, like the map at a different location, ViKiNG deviates from its original path (magenta) and reaches the goal by following the learned heuristic (blue).

upon overlaying this trajectory on the invalid map, we find that the learned heuristic indeed encourages the robot to follow the curvature of the road, but this path is still successful because it corresponds to open space.

4.5.4 *A Disoriented ViKiNG*

Finally, we analyze the effects of *disabling* the geographic hints and GPS localization on the goal-seeking performance of ViKiNG. Towards this, we run two variants of our algorithm:

No Overhead Image: We provide the robot with GPS, but no satellite images. To accommodate this, we use a simple ℓ_2 heuristic $h_{\text{GPS}}(x_w, x_g, x_t) = \|x_g - x_w\|$.

No GPS: The robot does not have access to GPS or satellite images. To accommodate this, we remove the heuristic h from ViKiNG-A*, making it an uninformed search algorithm.

Figure 24 summarizes the path taken by the robot, distance traversed, and time taken. When we disable the overhead hints and only use h_{GPS}, ViKiNG-A* can still reach the destination, but takes significantly longer to do so, initially exploring a dead-end path that it then has to back out of. That said, this experiment also illustrates the ability of ViKiNG-A* to handle less useful heuristics: while the path is significantly longer, the method is still able to eventually reach the destination, and in some sense the mistakes the method makes are to be expected of any system that has no prior map information. If

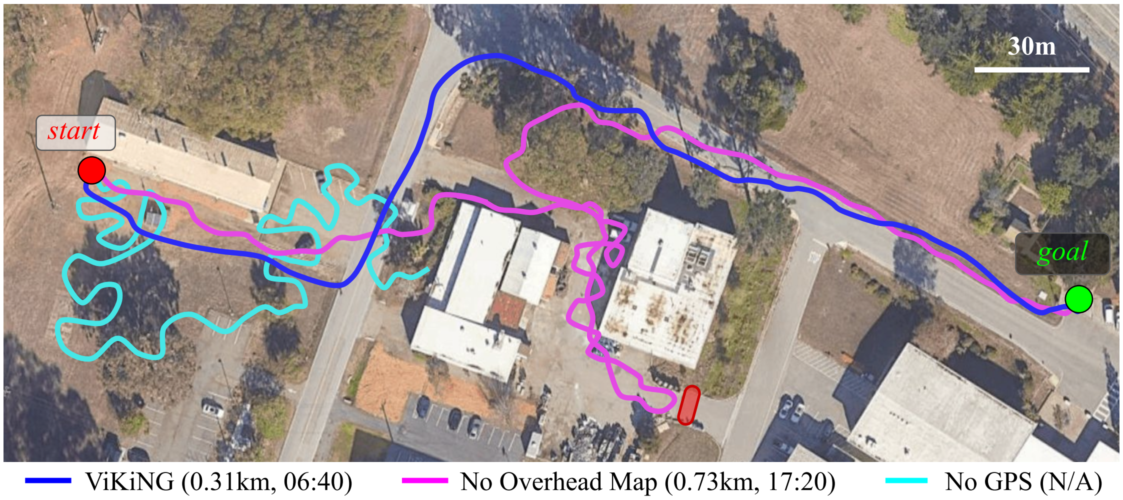


Figure 24: Ablations of ViKiNG by withholding geographic hints. ViKiNG without overhead images (magenta) acts greedily, driving close to buildings, gets caught into a cul-de-sac and eventually reaches the goal 2.6× slower that ViKiNG with access to satellite images (blue), which avoids the building cluster by following a smoother dirt path. Search without GPS (cyan) performs uninformed exploration and is unable to reach the goal in over 30 minutes.

we remove GPS as well, ViKiNG-A* corresponds to a Dijkstra-like uninformed search (resembling RECON [238]). In this case, the robot searches its environment without any guidance and is unable to reach the goal in over 30 minutes.

4.6 DISCUSSION

We proposed a method for efficiently learning vision-based navigation in *previously unseen* environments at a kilometer-scale. Our key insight is that effectively leveraging a small amount of geographic knowledge in a learning-based framework can provide strong regularities that enable robots to navigate to distant goals. We find that incorporating geographic hints as goal-directed heuristics for planning enables emergent preferences such as following roads or hiking trails. Additionally, ViKiNG only uses the hints for biasing the high-level search; the learned control policy at the lower-level relies solely on egocentric image observations, and is thus robust to imperfect hints. While we only use overhead images in our experiments, an existing avenue for future work is to explore how such a system could use other information sources, including paper maps or textual instructions, which can be incorporated into our contrastive objective.

Acknowledgments

This research was partially supported by DARPA Assured Autonomy, ARL DCIST CRA W911NF-17-2-0181, DARPA RACER, and Toyota Research Institute. The authors would like to thank Blazej Osinski, Dieter Fox, Tambet Matiisen, Brian Ichter, and Katie Kang for useful discussions.

5

OFFLINE REINFORCEMENT LEARNING FOR VISUAL NAVIGATION

Synopsis

This chapter proposes ReViND, the first robotic navigation system that can leverage previously collected data to optimize user-specified reward functions in the real-world using offline reinforcement learning. In contrast with the behavior cloning-based objectives proposed in ViNG, reinforcement learning can enable robots to navigate to distant goals while optimizing user-specified reward functions, including preferences for following lanes, staying on paved paths, or avoiding freshly mowed grass. However, online learning from trial-and-error for real-world robots is logistically challenging, and methods that instead can utilize existing datasets of robotic navigation data could be significantly more scalable and enable broader generalization. We evaluate our system for off-road navigation without any additional data collection or fine-tuning, and show that it can navigate to distant goals using only offline training from this dataset, and exhibit behaviors that qualitatively differ based on the user-specified reward function.

5.1 INTRODUCTION

Robotic navigation approaches aim to enable robots to navigate to user-specified goals in known and unknown environments. The *geometric* approach to this problem involves using a geometric map of the environment to plan a collision-free path towards the goal. The *learning-based* approach to this problem involves training policies by associating new inputs with prior navigational experience, typically through imitation learning (IL) or reinforcement learning (RL). In many practical applications, the goal is not merely to *reach* a particular destination, but to do so while maximizing some desired *utility measure*, which could include obeying the rules of the road, staying in a bike lane, maintaining safety, or even more esoteric goals such as remaining in direct sunlight for a solar-powered

Shah†, Bhorkar†, Leen, Kostrikov, Rhinehart, Levine, "Offline Reinforcement Learning for Visual Navigation", in Annual Conference on Robot Learning (CoRL) 2022
Project website: sites.google.com/view/revind

vehicle. In these cases, neither IL nor geometric approaches alone would suffice without accurate reconstructions of the environment or task-specific expert demonstrations, which may be difficult to obtain. RL can address these challenges, but prior work on applying RL to robotic navigation relies on infeasible amounts of online data collection, or requires high-fidelity simulators for simulation to real world transfer [283, 121]. Is there a practical RL paradigm that can solve this challenge directly from real-world data?

RL from offline datasets [155] can address this challenge by learning policies from a prior dataset of trajectories and associated reward labels. Given a previously collected diverse dataset of navigational trajectories, it is possible to relabel that dataset post-hoc with reward labels as desired, train a policy that maximizes this reward function, and deploy it in the real world. Since this approach can leverage large datasets, it may lead to significantly better generalization [128] than methods that require much more tightly curated data, such as imitation learning methods. However, end-to-end trained "flat" RL policies tend to perform poorly for long-horizon tasks [10, 65, 241]. How can we design a system for learning control policies from large datasets that can be immediately deployed onto a mobile robot?

In this paper, we describe a robotic learning system that performs visual navigation to distant goals (e.g. 100s meters away) while also incorporating user-specified reward objectives. Our system consists of two parts: (i) an offline Q-learning algorithm [137] that can incorporate the desired preferences in the learned Q-function and trains a policy operating directly on raw visual observations, and (ii) a topological representation of the environment for planning, where nodes are represented by the raw visual observations and the connectivity between them is described by the learned value function (see system overview in Fig. 25). While the Q-function alone may only be sufficient to learn accurate navigational strategies over short horizons, composing it with *planning* allows scaling to large environments by searching for a plan that maximizes the desired objective at a coarse level. The low-level policy derived from the Q-function is subsequently used to navigate between the nodes, maximizing the desired objective at both levels.

The primary contribution of this work is ReViND, a robotic system for **Re**inforcement learning for **Vi**sual **N**avigation from prior **D**ata that can act in real-world environments, adopt behavior that maximizes the user-specified reward functions, and reach distant goals, by combining planning and Q-learning. We demonstrate that ReViND can incorporate high-level rewards, such as staying on pavements or driving in sunlight and reach goals in complex environments over hundreds of meters. ReViND is pre-trained on 30 hours of publicly available data [238] and is deployed in a novel, visually similar environment *without* any on-policy data collection or fine-tuning. To the best of our knowledge, this is the first demonstration of offline RL for real-world navigation utilizing only publicly available datasets. Our experiments show that ReViND demonstrates diverse qualitative behaviors by tweaking the reward objectives while outperforming policies trained with IL and model-free RL.

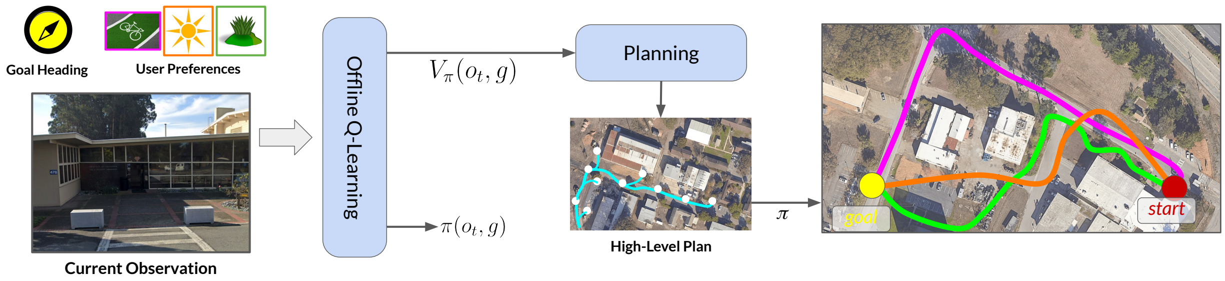


Figure 25: Long-range RL with ReViND: We use Implicit Q-learning to learn a goal-conditioned policy π and it's corresponding value function V_π from an offline dataset of interactions and user preferences, encoded as rewards. We then create a topological graph using $-V_\pi$ as the pairwise "distance function". The minimum-cost path to the goal in this graph is the desired reward-maximizing path to the goal, resulting in varied behaviors such as goal-reaching while driving on the grass, or following a bike lane.

5.2 RELATED WORK

Reinforcement learning (RL) approaches to navigation have shown great success in learning from large-scale data [171, 283]. A challenge with such methods is that RL algorithms can require a large amount of online experience (e.g., millions or even billions of trials) [283]. A method that requires a million 1-minute episodes would take more than 1.5 years of nonstop real-world collection, making it poorly suited for learning from scratch directly in the real world. Therefore, such methods typically require simulation, transfer, and other additional components [28, 136, 7]. An alternative for encoding specific user preferences into a learning-based method is to employ imitation learning (following Chapters 2–4). While imitation learning can enable a user to define their desired behavior through the demonstrations, such demonstrations are time-consuming to gather, and must be recollected for each new reward function. In contrast, our offline RL method utilizes previously collected datasets, which we show is practical for real-world robots, and relabels the same dataset with different reward functions, which means no reward-specific data collection is needed.

Prior offline RL work has proposed a number of algorithms that can utilize previously collected data [77, 148, 155, 77, 186, 145, 138, 137]. Our goal is not to develop a new offline RL algorithm, but rather to explore their application to robotic navigation tasks. Most prior robotics applications of such methods include multi-task learning for tabletop manipulation [126, 248, 170, 151, 32]. In this chapter, specifically explore how a single dataset can be reused to enable long-horizon navigation with different user preferences. To that end, we combine our approach with graph-based search to reach distant goals, which we show significantly improves over direct use of the learned policy, and utilize the same exact data to optimize different reward functions. While prior work has also explored the use of offline data with varying reward functions [123, 125], we address significantly longer horizon tasks by incorporating model-free RL and graph search.

Our use of graph search in combination with RL parallels prior work that integrates planning into supervised skill learning methods (Chapter 2) and goal-conditioned reinforcement learning [65, 69, 189]. However, our method differs from these works in two ways. First, while these prior works use the value function to estimate the temporal distance between pairs of nodes in the graph, we specifically explore using divese objectives. More importantly, we use offline RL, whereas prior work uses either supervised regression for distances, or online RL. Our goal is not to develop a new offline RL algorithm, but to explore it's application to robotic navigation tasks by building a learning-based system for long-horizon planning. To our knowledge, our work is the first to combine topological graphs with RL for arbitrary reward functions, and the first to combine them with offline RL.

5.3 OFFLINE REINFORCEMENT LEARNING FOR LONG-HORIZON NAVIGATION

Our system combines offline learning of reward-specific value functions with topological planning over a graph constructed from prior experience in a given environment, so as to enable a robot to navigate to distant goals while maximizing user-specified rewards. The learned value function is used not only to supervise a local policy that chooses reward-maximizing actions, but also to evaluate edge costs on a graph constructed from past experience. The graph is then used to plan a path, and the policy is used to execute the action to reach the first subgoal on that path. Structurally, this resembles SoRB [65], but with two critical changes: learning "offline" value functions from prior data, and the ability to handle diverse reward functions beyond simple goal-reaching.

5.3.1 *Problem Statement and Assumptions*

The robot's task is defined in the context of a goal-conditioned Markov decision process, with state observations $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, and goals $g \in \mathcal{G}$. The robot receives a reward $r(s_t)$ at each time step t, which depends on the degree to which it is satisfying user preferences (e.g., staying on the graph). The objective can be expressed as maximizing the total reward of the robot's executed path, since the reward accounts for both the desired utility and goal reaching. The state observations consist of RGB images from the robot's forward-facing camera and a 2D GPS coordinate, the actions are 2D steering and throttle commands, the goal is a 2D GPS coordinate expressed in the robot's frame of reference. In this setting, reinforcement learning methods will learn policies of the form $\pi(a_t|s_t, g_t)$, though our approach will not command the final task goal g_t directly, but instead will use a planning method to determine intermediate subgoals, which in practice makes it significantly easier to reach distant goals. To enable this sort of planning, we make an additional assumption that parallels prior work on combining RL with graph search [65, 241]: we assume that the robot has access to prior experience from the current environment that it can use to build a topological graph that describes its connectivity. Intuitively, this corresponds to a kind of "mental map" that describes which landmarks are reachable from which other landmarks. Importantly, we do *not* assume that this graph is manually constructed or provided: the algorithm constructs the graph automatically using an uncurated set of observations recorded from prior drive-throughs of the environment. In our experiment, these traversals are done via teleoperation, though they could also be performed via autonomous exploration, and our method could be extended to handle unseen environments by integrating the exploration procedures discussed in prior work [238].

5.3.2 *Reinforcement Learning from Offline Data*

Offline RL algorithms learn policies from static datasets. In our implementation we use implicit Q-learning (IQL) [137], though our approach is compatible with any value-based offline RL algorithm. We summarize offline RL in general and IQL specifically in this section. Given a dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i) \mid i = 1 \ldots N\}$, the goal of offline RL is to learn a policy that optimizes the sum of discounted future rewards without any additional interactions with the environment. IQL involves fitting two neural networks, Q_θ and V_ψ, where $Q_\theta(s, a, g)$ approximates the Q-function of an *implicit* policy that maximizes the previous Q-function, and V_ψ represents the corresponding value function. The Q-function is updated by minimizing squared error against the next time step value function, with the objective

$$L(\theta) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}, g \sim p(g|s)}[(\gamma V_\psi(s', g) + r(s, a, g) - Q_\theta(s, a, g))^2],$$

where $p(g|s)$ is a goal distribution, which we will discuss later. The value function $V_\psi(s, g)$ should be trained to correspond to $Q_\phi(s, a, g)$ for the optimal action a that maximizes the value at s, but directly computing $\max_a Q_\phi(s, a, g)$ is likely to select an "adversarial" out-of-distribution action that leads to erroneously large values, since the static dataset does not permit $Q_\phi(s, a, g)$ to be trained on all possible actions [137, 155, 145]. Therefore, IQL employs an *implicit* expectile update, with a loss function given by

$$L(\psi) = \mathbb{E}_{(s,a) \sim \mathcal{D}, g \sim p(g|s)}[L_2^\tau(Q_\theta(s, a, g) - V_\psi(s', g))],$$

where $L_2^\tau(u) = |\tau - \mathbb{1}(u < 0)|u^2$. This can be shown to approximate the maximum over *in-distribution* actions [137], but does not require ever querying out-of-sample actions during training. To instantiate this method, it remains only to select $p(g|s)$.

Goal relabeling. The IQL algorithm is not goal-conditioned [137], and the dataset was not collected with a goal-reaching policy, so the goals must be selected post-hoc with some sort of *relabeling strategy*. While a variety of relabeling strategies have been proposed in prior work [122, 10, 32, 65, 66], we follow prior work on offline RL for goal-reaching [32] and simply set the goal to states that are observed in the dataset in the same trajectory at time steps subsequent to a given sample s_i. In our implementation, we select this time step at random between 10 and 70 time steps after s_i (the total trajectory lengths are typically around 80 steps). Algorithm 6 outlines pseudocode for training the Q-function with IQL.

Long-horizon control. Instead of directly using the policy learned with IQL, in this paper we use the IQL value function to obtain edge costs for a graph used for topological planing. The standard IQL method directly extracts a reactive policy from the Q-function. However, we found that in the real world, this approach was unable to reach goals farther than 20m, or 80 time steps. A deeper analysis of the system revealed that, while the policy and values learned by the IQL agent are valid over shorter horizons, they degrade rapidly as the horizon increases. This is not surprising, because like all value-based methods, IQL assumes that s represents a Markovian state. But this assumption becomes increasingly violated for long-horizon tasks with first-person images: while goals that are within line of sight of the robot are relatively simple, goals that require navigating around obstacles tend to fail if using the policy directly. In the next subsection, we will discuss how we can use a topological graph as a sort of "nonparametric memory" of the environment to alleviate this challenge, enabling our method to reach distant goals.

5.3.3 *Long-Horizon Reward Maximization with a Topological Graph*

To enable long-horizon navigation, we combine the value function learned via offline RL with a topological graph built from prior observations in a given environment. As discussed previously, we assume that the robot has a limited amount of prior experience in the test environment that can be used to build a "mental map," corresponding to a graph where nodes are observations and edges represent the cumulative reward the robot will accumulate as it travels from one node to another. Note that this graph is topological rather than geometric: the nodes are image observations, and the connectivity is determined by the learned value function. We do not use the data from the test environment to finetune the value functions, only to construct the graph.

The graph $\mathcal{G}$ is constructed in the same way as prior work on graph-based navigation (see 2 for the closest prior method): each state observation s_i in the test environment corresponds to a node n_i, and each edge e_{ij} receives a cost corresponding to $C(e_{ij}) = -V_\psi(s_i, s_j)$.[1] We further filter these edges based on the GPS coordinates of the nodes to eliminate *wormholes* arising due to optimistic value estimates. For more details on how the graph is constructed, please see Appendix C.3. Given an overall task goal, we add it to the graph as an additional node n_G, along with a node representing the robot's current state, and then use Dijkstra's algorithm to compute the shortest path with these edge

[1] In our implementation, goals are defined only in terms of GPS coordinates, so technically, the second argument is only the GPS coordinate of s_j, which we found to be sufficient. Extending the method to use the full image observation is a simple modification.

costs. We then use the policy learned via offline RL to navigate to the first node along this path. Algorithm 7 outlines pseudocode for this procedure.

In our implementation, we use goal-conditioned reward functions of this form:

$$R(s_t, a_t, g) = \begin{cases} -k_t(s_t) & \forall s_t \neq g \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

where $k_t(s_t) > 0$ is always positive to ensure that the planner actually reaches the goal. **Proposition 3.1** *If we recover the optimal value function $V^*(s, s')$ for short-horizon goals s' (relative to s), and $\mathcal{G} = \mathcal{S}$ (all states exist in the graph), and the MDP is deterministic with $\gamma = 1$, then finding the minimum-cost path in the graph $\mathcal{G}$ with edge-weights $-V^*(s, s')$ recovers the optimal path, that is, a policy π that maximizes $V^*(s, g)$.*

Proof (sketch): The Bellman equation can be used to write the cost of the minimal-cost path in the graph with edge-weights $-V(s, s')$: $J^*(s, g) = \min_{s'}[-V(s, s') + J^*(s', g)] = -\max_{s'}[V(s, s') - J^*(s', g)]$. We can further expand $V(s, s')$ into a sum of rewards induced by the policy π and then rearrange the terms to obtain a similar optimality equation for V^* that demonstrates that $J^*(s, g) = -V^*(s, g)$. While the above proposition makes strong assumptions, it provides some degree of confidence that our proposed method is *correct* and *consistent*. We present further analysis of this proposition in Appendix C.1.

Algorithm 6 Training ReViND

```
Initialize parameters $\psi$, $\theta$, $\hat{\theta}$, $\phi$.
for each gradient step do
    Sample a mini-batch $\{(s_i, a_i, r_i, s_i')\}$
    for each sample do
        $T \leftarrow T \in D \mid s_i \in T$
        $g_i \leftarrow$ SampleGoal$(T, s_i)$
        $s_i, s_i' \leftarrow$ Relabel$(s_i, s_i', g_i)$
        $r_i \leftarrow$ Reward$(s_i, g_i)$
    end for
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi L_V(\psi)$
    $\theta \leftarrow \theta - \lambda_Q \nabla_\theta L_Q(\theta)$
    $\hat{\theta} \leftarrow (1 - \alpha)\hat{\theta} + \alpha\theta$
end for
```

Algorithm 7 Deploying ReViND

```
Inputs: current observation obs $:= \{\text{img}, x\}$, set of past observations $\mathcal{N} := \{n_1, \ldots, n_m\}$, IQL agent $\{Q, V, \pi\}$, goal node $n_G \in \mathcal{N}$
$\mathcal{G} \leftarrow$ ConstructGraph$(\mathcal{N}, V)$
while not IsClose(obs, $n_G$) do
    UpdateGraph(obs)
    $w_1, \ldots, w_k \leftarrow$ DijkstraSearch(obs, $n_G$)
    for $t = 1, \ldots, H$ do
        goal vector = GetRelative$(x, w_1)$
        RunPolicy(img, goal)        ▷ runs on robot
        obs $\leftarrow$ next observation
    end for
end while
```

5.4 System Evaluation

We now describe our system and experiments that we use to evaluate ReViND in real-world environments with a variety of utility functions. Our experiments evaluate ReViND's ability to incorporate diverse objectives and learn customizable behavior for long-horizon navigation, and compare it alternative methods for learning navigational skills from offline datasets.

5.4.1 *Mobile Robot Platform*

We implement ReViND on a Clearpath Jackal UGV platform (see Fig. 25). The sensor suite consists of a 6-DoF IMU, GPS for approximate localization, and wheel encoders to estimate local odometry. The robot observes the environment using a forward-facing 170° field-of-view RGB camera. Compute is provided by an NVIDIA Jetson TX2 computer, with the RL controller running on-board. Our method uses only the images from the on-board camera and unfiltered GPS measurements.

5.4.2 *Offline Trajectory Dataset and Reward Labeling*

The ability to utilize offline datasets enables ReViND to learn navigation behavior directly from existing datasets — which may be expert tele-operated or collected via an autonomous exploration policy — without collecting *any* new data. We demonstrate that ReViND can learn behaviors from a small offline dataset and generalize to a variety of previously unseen, *visually similar* environments including grasslands, forests and suburban neighborhoods. To emphasize this, we train ReViND using 30 hours of publicly available robot trajectories collected using a randomized data collection procedure in an office park [238]. Expanding this training dataset to include more diverse scenes can help extend these results to alternate applications (e.g. indoors).

To utilize this data with our method, we "relabel" it with several different reward labels corresponding to diverse behaviors: simple shortest-path goal-reaching, driving in the sun (to emulate a solar-powered vehicle that needs sunlight), driving on grass (to stay off the road), and driving on the pavement (to stay off the grass). We generate these labels by different mechanisms — either by manually labeling them, by using a learned reward classifier network, or automatically, by exploiting pixel-level patterns (e.g., in the color space). We implement these rewards via additive bonus to the negative rewards which corresponds to reducing the penalty for traversing these areas. For more details, see Appendix C.2. As discussed in Sec. 5.3.2, we use IQL to learn the value functions and policies for each task.

5.4.3 *Learning Varied Behaviors with ReViND*

We now evaluate our method both in terms of its ability to tailor the navigational strategy to the provided reward, and in terms of how it compares to prior approaches and baselines. We test ReViND in five suburban environments for a large number of goal-reaching tasks (see Appendix C.6). While these environments are visually similar to the offline training data, they exhibit dynamic elements such as moving obstacles, automobiles, and changes in the appearance of the environment across the seasons. In each evaluation environment, we construct a topological graph by manually driving the robot and collecting visual and GPS observations. The nodes of this graph are obtained by sub-sampling these

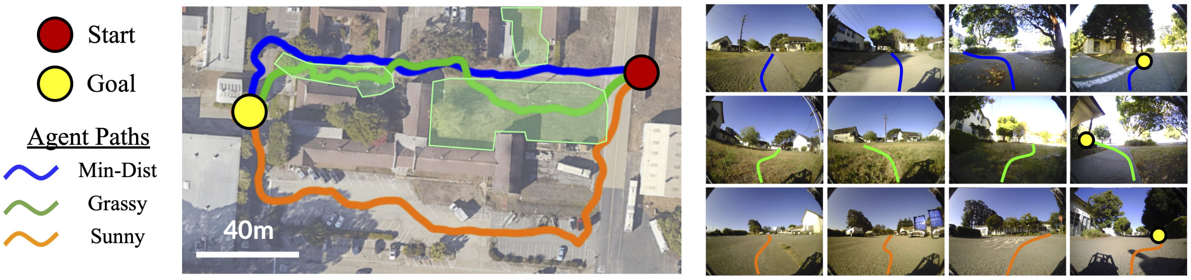


Figure 26: Comparison of policies for different reward functions learned by ReViND. Left: an overhead map (not available to the method), with grassy areas indicated with green shading. Note that the policy for the "sunny" reward chooses a significantly different path through a concrete parking lot without tree cover, while the policy for the "grassy" reward takes frequent detours to drive on lawns. Right: first person images during each traversal, with the chosen path indicated with colored lines.

observations, such that they are 10–30m apart, and the edge connectivity is determined by the corresponding value estimates. Note that the Q-function is *not updated* with this data, it is only used to build the graph. Once the graph is constructed, the robot is tasked with reaching a goal location, where it follows Alg. 7 to search for a path through the graph, and then executes it via the learned policy. Fig. 26 shows the paths taken by different policies for a specific start-goal pair. The overhead image is not available to ReViND and is only provided for illustration.

Our results show that utilizing value functions for different rewards from ReViND leads to significantly different paths through the environment. For example, the "sunny" reward function causes a large detour through a parking lot without tree cover, while the "grassy" reward causes frequent detours to drive on lawns. All of the policies successfully avoid obstacles and collisions and successfully reach the goal. In Table 7 we provide a quantitative summary of the behavior of the method for each reward function, showing success weighted by path length (SPL, which corresponds to an optimality measure that awards higher scores to successful runs with the shortest route length), the average value of the grass reward, and the average value of sun reward for trials corresponding to each reward function (note that these rewards are normalized to maximum of 1). As expected, we see that the values of these metrics strongly covary with the commanded reward.

Agent Utility	SPL	$\mathbb{E}R_{\text{grass}}$	$\mathbb{E}R_{\text{sun}}$
R_{dist}	**0.87**	0.16	0.61
$R_{\text{dist}} + R_{\text{grass}}$	0.84	**0.86**	0.39
$R_{\text{dist}} + R_{\text{sun}}$	0.64	0.05	**0.68**

Table 7: ReViND learns diverse behaviors that maximize the desired utility.

Next, we compare ReViND to four baselines, each trained on the same offline dataset. These approaches represent natural points of comparison for our method, and include

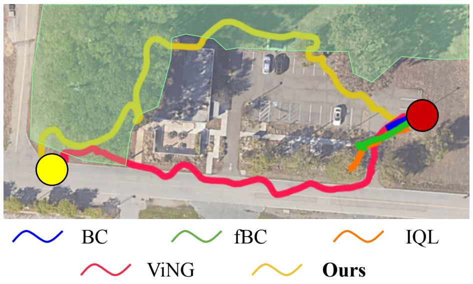


Figure 27: Qualitatively, only ReViND reaches the goal while prioritizing grassy terrain (shaded green).

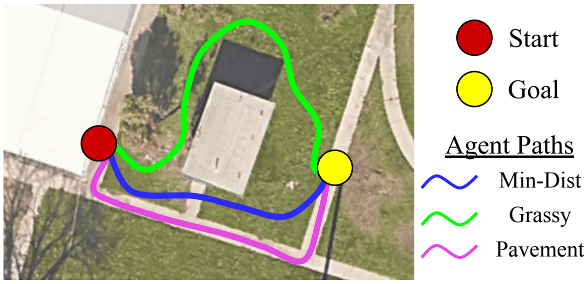


Figure 28: ReViND takes different paths through the environment for different reward functions.

Method	Uses Graph?	Easy (<50m)		Medium (50–150m)		Hard (150–500m)	
		Success	$\mathbb{E}\mathbb{1}_{\text{grass}}$	Success	$\mathbb{E}\mathbb{1}_{\text{grass}}$	Success	$\mathbb{E}\mathbb{1}_{\text{grass}}$
Behavior Cloning	No	1/5	0.08	0/5	0.04	0/5	0.12
Filtered BC	No	3/5	0.29	0/5	0.08	0/5	0.12
IQL [137]	No	3/5	0.37	1/5	0.29	0/5	0.16
ViNG [241]	Yes	**5/5**	0.07	**4/5**	0.09	3/5	0.14
Filtered BC + Graph	Yes	**5/5**	0.24	**4/5**	0.15	3/5	0.19
ReViND (Ours)	Yes	**5/5**	**0.47**	**4/5**	**0.84**	**4/5**	**0.78**

Table 8: Success rates and utility maximization for the task of navigation in grassy regions (R_{grass}).

prior imitation learning and RL methods, as well as a prior graph-based method that does not use RL. Since our approach is (to our knowledge) the first to combine RL with arbitrary rewards and topological graph search, no prior approach supports both graphs and arbitrary rewards. All methods have access to egocentric images and GPS, and command future waypoints to the robot.

Behavioral Cloning (BC): A goal-conditioned imitation policy that maps images and goals to control actions [44]. *This baseline does not incorporate reward information.*

Filtered BC (fBC): A similar goal-conditioned BC policy that incorporates reward information by filtering the training data, picking only trajectories with the top 50% aggregate rewards [37].

ViNG: A graph-based navigation system that combines a goal-conditioned BC policy and distance function with a topological graph [241]. *This baseline does not incorporate reward information.*

IQL: A baseline that uses only the learned Q-function, without a topological graph [137].

Fig. 27 shows the qualitative behavior exhibited by the different systems for maximizing the "grassy" reward function. IQL and Filtered BC can incorporate the reward function into the policy, but since they rely entirely on a reactive policy for navigation, they are unable to determine how to navigate toward the goal, and exhibit meainingless bee-lining behavior. Using a graph search to find a minimum distance path, ViNG can reach the goal, but does not satisfy the reward function. Only ReViND is successful in navigating to the goal while taking a short detour that maximizes the desired objective, demonstrating affinity to grassy terrains.

We provide a quantitative evaluation of these methods in Table 8 and Appendix C.4, showing the average distance traveled by each method over all test trials prior to disengagement, as well as the average value of the utilities. We see that non-RL methods are unable to take into account the task reward, and simply aim to reach the task goal, which leads to suboptimal utility. We can take reward into account either using RL, or by filtering BC to imitate only the high-reward trajectories. In easier environments, we see that both fBC and IQL can learn reward-maximizing behavior. However, both the RL and BC flat policies suffer sharp drops in performance as the distance to goal increases. The addition of a graph greatly helps improve performance. Here again, we notice that offline RL (ReViND), which uses Q-learning to optimize the reward, consistently outperforms filtering-based approaches (fBC-graph) — this confirms that reward information is important for respecting the user's preferences, and that offline RL is more effective at this than filtering.

The biggest failure mode for current offline RL and IL methods in our task is their inability to reach distant goals. BC, fBC and IQL consistently fail to reach goals beyond 15-20m away, due to challenges in learning a useful policy from offline data — these *flat* baseline policies often demonstrate *bee-lining* behavior, driving straight to the goal, which often leads to collisions.

5.5 DISCUSSION

We presented ReViND, a robotic navigation system that uses offline reinforcement learning in combination with graph search to reach distant goals while optimizing user preferences. We showed that ReViND can be trained on a navigational dataset collected in prior work, in combination with reward labeling, to exhibit qualitatively distinct behaviors. Our experiments show that ReViND can generalize to novel, visually similar environments, and is responsive to the user preferences, significantly outperforming prior methods that either do not utilize high-level planning, or utilize graphs without RL and therefore do not support reward specification. We hope that our work will provide a step towards robotic learning methods that routinely reuse existing data, while still accomplishing new tasks and optimizing user preferences. Such methods can exhibit effective generalization in the real world through their ability to incorporate existing diverse datasets, while also flexibly solving new tasks, so long as the specified reward functions are valid in the novel environments and tasks.

Part II

CROSS-EMBODIMENT *ROBOT FOUNDATION MODELS*

6

A GENERAL NAVIGATION MODEL TO DRIVE ANY ROBOT

Synopsis

Learning provides a powerful tool for vision-based navigation, as showcased in Part I, but the capabilities of learning-based policies are constrained by limited training data. If we could combine data from all available sources, including multiple kinds of robots, we could train more powerful navigation models. In this chapter, we study how a *general* goal-conditioned model for vision-based navigation can be trained on data obtained from many distinct but structurally similar robots, and enable broad generalization across environments and embodiments. We analyze the necessary design decisions for effective data sharing across robots, including the use of temporal context and standardized action spaces, and demonstrate that an *omnipolicy* trained from heterogeneous datasets outperforms policies trained on any single dataset. We curate 60 hours of navigation trajectories from 6 distinct robots, and deploy the trained GNM on a range of new robots, including an underactuated quadrotor. We find that training on diverse data leads to robustness against degradation in sensing and actuation.

6.1 INTRODUCTION

Machine learning methods have enabled broad generalization with real-world applicability in natural language processing [206], visual perception [53, 26, 85], and other domains [209, 38] by leveraging Internet-scale data. Such generalization typically requires learning general patterns from diverse datasets, which are usually collected once and then reused for various purposes. Such large-scale models also support the ability to be adapted for new tasks by reusing the representations learned from broader, larger, and more general datasets, for example by or zero-shot transfer [39, 239, 280], or fine-tuning

Shah†, Sridhar†, Bhorkar, Hirose, Levine, "GNM: A General Navigation Model to Drive Any Robot", in IEEE International Conference on Robotics and Automation (ICRA) 2023
Project website: general-navigation-models.github.io

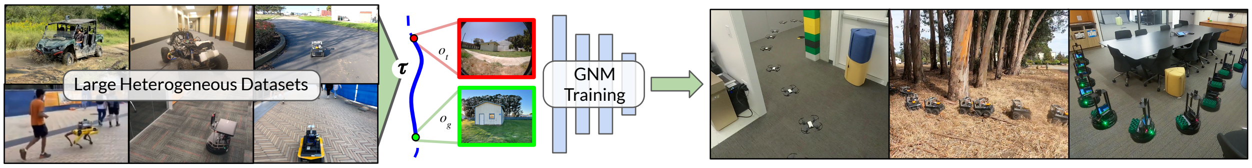


Figure 29: A general navigation model to drive any robot. By training on diverse, heterogeneous datasets, a single "omnipolicy" can control a variety of robots in challenging environments, including *new* robots, without any robot-specific data collection.

on target-domain data. Although this paradigm has been very successful, it is difficult to apply in robotics due to the sheer diversity of environments and platforms across researchers. Control policies learned end-to-end usually require separate data collection for each robotic platform, leading to "fragmentation" in progress, where every researcher works with their own robot-specific dataset and policies, making it infeasible to accumulate large enough datasets. Can we overcome this challenge by training models on more general and reusable cross-robot datasets?

We study this question in the context of visual navigation, where heterogeneity between robots might include different camera hardware, viewpoints, dynamics, and more broadly, embodiments, but where the over-arching navigation objective looks similar irrespective of these differences. A wheeled robot, quadruped, or a drone all have the same abstract objectives: to explore the environment, plan a path to the goal, and avoid collisions. Leveraging this shared abstraction across robots and training a general navigational *omnipolicy* from large-scale data could enable broad generalization to novel environments, unseen sensor parameters (e.g., camera intrinsics and extrinsics), and new robot configurations.

In this paper, we propose to take a step towards this kind of data sharing by training an embodiment-agnostic general navigation model (GNM) from an aggregated multi-robot dataset. The primary contribution of our work is a framework for training a *general* omnipolicy from multi-robot datasets, with empirical evidence that such an omnipolicy can effectively learn from heterogeneous datasets and generalize to novel robot platforms. To facilitate this, we aggregate a large heterogeneous dataset of navigation trajectories collected across 6 robots, spanning 60 hours of interactions in challenging indoor and outdoor environments. We train the GNM on this dataset and deploy it on 4 distinct robot platforms, including 2 new robots. We show that a single learned policy can be used across multiple robots to perform goal-reaching in challenging indoor and outdoor environments, outperforming policies trained with any single dataset. We also report robustness to degradation in camera parameters, tire damage, and other gradual changes that the robot may experience over its lifetime.

We have publicly released the trained GNM policy, code used to train and deploy our models on various popular robot platforms, as well as the dataset used to train these models at our project page. We hope that this represents a step towards both general-purpose multi-robot datasets and general-purpose *visual navigational models* that can be deployed on a wide range of robots — similar to how practitioners currently use pre-trained models in vision and language, such models could constitute pre-trained backbones for visual navigation.

6.2 RELATED WORK

Learning from large, diverse robotic datasets has been studied for various robotic applications where data sharing across *similar* robots helps scale learning to challenging environments [54, 50, 297]. However, for applications such as ground or aerial navigation, with different sensors and robot dynamics, current approaches tend to rely on learning from small datasets which are only representative of a single robotic platform. Our paper proposes learning navigation behavior from heterogeneous robot datasets, collected across multiple embodiments.

Our work is closely related to transfer learning, where the objective is to train policies that transfer across domains, such as across dynamics [298, 200, 71], environments [144], morphologies [88, 107, 127], viewpoints [218], and embodiments [97]. Our focus is not on designing specific domain adaptation algorithms or hand-engineered augmentations [97] for transfer, but rather studying how direct generalization of simple, high-capacity models trained on real-world data can provide a path to broadly applicable navigational policies. Towards this, our work is also closely related to DroNet [163], which imitates expert on-road driving data to control a quadrotor. We take this paradigm one step further, showing that we can train *goal-conditioned* policies on data from *multiple* robots and control new ones, including a quadrotor.

Prior work has also explored learning of visual representations or end-to-end policies from passive data, such as YouTube videos, which can be scaled up massively without real-world data collection [27, 89, 187, 207]. We explore a complementary direction, studying how readily available on-robot data (also passive) can lead to generalizable policies. This is particularly relevant for navigation, where data is plentiful, and trajectories from multiple robots can directly train a policy, as opposed to two-stage methods that use Internet data for representation learning followed by in-domain adaptation.

Following the findings in Part I, we use a combination of topological graphs for high-level planning and image-goal policies for low-level control, which gives us an efficient way to scale reactive policies for long-range navigation [179, 65]. We show that that our GNM can be coupled with such topological graphs to scale image-goal navigation to new robots.

	Dataset	Platform	Speed	Hrs.	Environment
1	GoStanford [96]	TurtleBot2	0.5m/s	14h	office
2	RECON [238]	Jackal	1m/s	25h	off-road
3	CoryHall [123]	RC Car	1.2m/s	2h	hallways
4	Berkeley [232]	Jackal	2m/s	4h	suburban
5	SCAND-S [129]	Spot	1.5m/s	8h	sidewalks
6	SCAND-J [129]	Jackal	2m/s	1h	sidewalks
7	Seattle [229]	Warthog	5m/s	1h	off-road
8	TartanDrive [269]	ATV	10m/s	5h	off-road
9	NeBula [3]	ATV	10m/s	10h	off-road
	Ours			70h	

Table 9: The GNM training dataset contains 70 hours of navigation data in diverse environments across 6 different robots.

6.3 MULTI-ROBOT TRAINING DATASET

Our aim is to train a *general* visual navigation model that can learn broadly applicable navigational affordances across a variety of distinct robotic systems. To facilitate such large-scale policy learning, we aggregated a heterogeneous dataset of navigation trajectories sourced from 8 datasets collected on robotic platforms with varying dynamics, sensors, and behaviors. The datasets contain a variety of challenging indoor and off-road environments (Table 9 and Fig. 29). We have publicly released this dataset on the project page.

The GNM dataset contains over 60 hours of real-world navigation trajectories: a combination of tele-operated and autonomous navigation behaviors collected across 6 distinct robotic platforms, including 4 commercially available platforms (TurtleBot, Clearpath Jackal, Warthog and Spot) and 2 custom platforms (Yamaha Viking ATV, RC Car). The trajectories contain widely varying robot dynamics and top speeds ranging between 0.2 and 10m/s, operating in a diverse set of environments (e.g., office buildings, hallways, suburban, off-road trails, university campus etc.).

To train navigation policies that can operate solely from egocentric visual observations, the dataset contains forward-facing RGB images paired with the robot's commanded actions and local odometry measurements. Each robot has different camera parameters, necessitating any successful policy to generalize across variations in camera pose and intrinsic parameters, though all platforms use the same type of sensor (monocular RGB camera). It is straightforward to further expand GNM by adding other datasets of relevant navigation behaviors [16, 25], or mix-and-match subsets of the dataset based on the desired application,

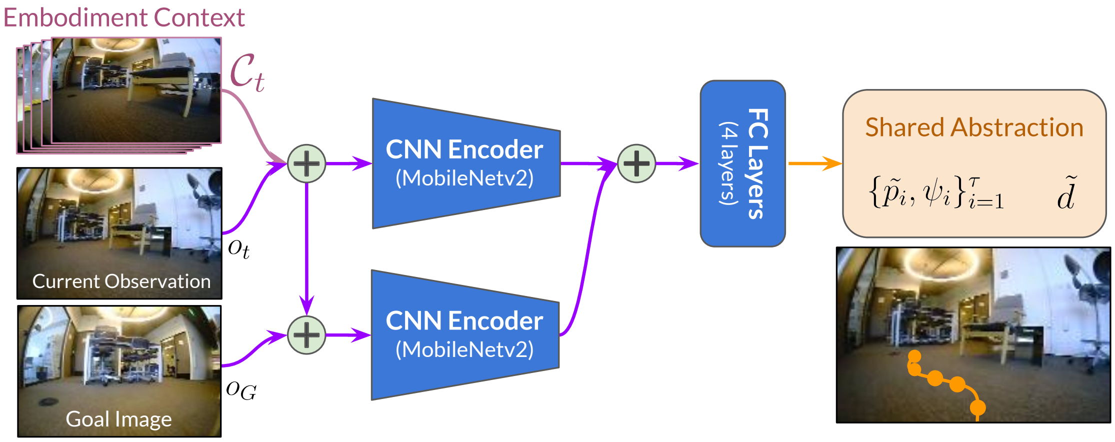


Figure 30: GNM architecture. We modify a typical goal-conditioned architecture (purple) by conditioning it on additional context from the target robot (pink) and making predictions in a shared, normalized action space (yellow).

6.4 TRAINING A GENERAL NAVIGATION MODEL

To study a common navigation task across robots and environments, we consider the problem of image-goal navigation [302], where a robot is tasked with navigating to a goal location G specified as an image observation o_G taken at G. Unlike PointGoal [7], GPS navigation, or semantic objectives [279], image-goal navigation is a general framework that does not rely on ground truth localization or semantic labels, and allows us to formulate a very general navigation task that can be trained with any visual navigation dataset. Our goal is to train a goal-reaching policy $\pi(o_t, o_G)$ that can navigate solely from egocentric visual observations. To provide a general task representation for this policy, we condition it on the desired goal o_G and integrate it into a navigational system based on topological graphs [96, 241, 250, 179].

Such systems have shown great navigation results in a variety of indoor and outdoor environments — what would it take to train such a policy *across* robots, with varying controllers, dynamics and sensor placements? We highlight two key ingredients in training multi-robot policies: (i) carefully choosing the right action representation that facilitates transfer across robots, and (ii) conditioning the policies on a "summary" vector that allows it to deduce the properties of the robot it is controlling, so different robots can exhibit different, valid capabilities. Although we found the particular design decisions described in this section to be important for good performance, as we discuss in our experiments (Sec. 6.5.3), we emphasize that the primary contribution of our work is *not* a novel learning algorithm, but an empirical demonstration that policies learned from heterogeneous datasets can generalize broadly to new environments and new robots.

6.4.1 *A Shared Abstraction Across Robots*

While the general task of navigation from egocentric images is common across robots, the specific inputs (camera observations) and outputs (actions, dynamics) can vary substantially: a TurtleBot is differential-drive, expects low-level velocity commands, and has a top speed of 0.5m/s, whereas an ATV uses Ackermann steering, expects throttle and steering commands, and drives up to 20× faster. Learning a common control policy that operates directly on these raw, unstructured outputs can be challenging due to these inconsistencies and high-variance outputs (e.g., speed $\in [0.2, 10]$m/s). This is further exacerbated when generalizing to new robots, where the policy might need to "guess" how fast it should move.

To this end, we propose using a shared abstraction to allow the goal-reaching policies to operate in a *transformed* action space that is consistent across robots, making the data points look "similar" and easier to learn common patterns from. In our experiments, we found this to be important to be able to learn from multiple datasets (see Sec. 6.5.3 for analysis). We use a combination of *relative* waypoints $p(x, y)$ and yaw change ψ as a mid-level action space. Labels for these actions can be obtained by using local odometry, which are easily available across datasets. Additionally, the policy also predicts the temporal distance to the goal d, as a measure of traversability, which is used by the navigation system to estimate the connectivity of the topological graph.

While this gives a shared action space across robots, we found that the varying dynamics (e.g., different top speeds) across robots can make it challenging for learning algorithms to learn a joint policy. To alleviate this, we propose using a *normalized* action space $\{\tilde{p}(x, y), \psi\}$, where $\tilde{p} := \frac{1}{\alpha} p$ is scaled by a robot-specific factor α corresponding to the top speed of the robot. The temporal distance $\tilde{d}$ is also estimated in this normalized scale. Given this abstract action space, a robot-specific controller can be used to (i) *unnormalize* the waypoints, and (ii) *track* them (e.g., PID, MPPI) to extract low-level commands (e.g., velocities or motor commands).

6.4.2 *Embodiment Context*

When deployed on an arbitrary robot, the policy must infer the capabilities of that particular robot. For instance, a TurtleBot can spin in-place but not go over bumps on the road, whereas an RC Car can easily traverse small bumps but has a limited turning radius. A simple way to provide such awareness to the policy is to condition it on hand-designed parameters that provide a concise "summary" of capabilities, such as its size, turning radius etc. Defining these parameters by hand presents a barrier to fast and easy deployment of the policy to new robots, and requires human intuition to identify and define a relevant set of parameters. Instead, we propose a simple and automatic approach: rather than manually defining parameters that fully identify the robot, we use a sequence of consecutive past observations from the robot's viewpoint to infer a learned *embodiment context* $\mathcal{C}_t$, and condition the learned policy on this context in addition to the observations. This context contains information about the robot's configuration and dynamics, which can be used to condition the behavior of the policy.

While this context may not contain *all* information to fully identify the robot, we hypothesize that it is sufficient to effectively control the robot. Our experiments show that the embodiment context allows the same policy to be deployed on novel robot configurations without designing any hand-engineered robot representation. We empirically evaluate different ways of providing context in Sec. 6.5.3 and find that the most effective representation is achieved by using a temporally consistent context $\mathcal{C}_t$ that conditions the policy on k consecutive past observations $\{o_{(t-k):(t-1)}\}$.

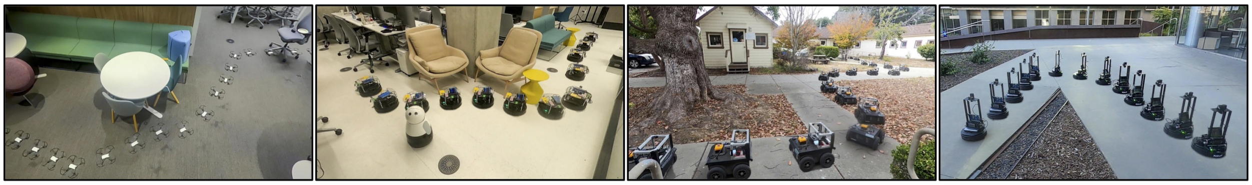


Figure 31: Depoying the GNM omnipolicy. We evaluate on 4 different robots in challenging indoor and outdoor environments.

6.4.3 *Implementation Details*

A combination of conditioning the policies on embodiment context and transforming the action space can allow a simple goal-reaching policy to be trained from heterogeneous datasets. It is important to note that the proposed modifications are orthogonal to the choice of downstream policy architecture and learning algorithm, and we could use different encoders or train with reinforcement learning.

ARCHITECTURE: We use a goal-conditioned policy architecture that takes as input the current observation o_t and goal observation o_G, and predicts *normalized* waypoints and distances. Additionally, we condition on temporal context $\mathcal{C}_t$, which is constructed by stacking the past $k = 5$ consecutive observations. Visual inputs to the network are provided as 85×64 RGB images for all observations. Following prior work [238, 185], we train context-conditioned representations by using separate MobileNetv2 encoders for (i) the current observation $\{o_t, \mathcal{C}_t\}$, and (ii) conditional goal observation, as shown in Fig. 30. The two embeddings are concatenated and passed through three fully-connected layers to two prediction heads: normalized temporal distance $\tilde{d}_t$ and a sequence of $\tau = 5$ normalized future waypoints $\{\tilde{p}_i, \psi_i\}_{i=1}^{\tau}$.

TRAINING: Following the procedure of Shah et. al. [241], we use a combination of image-goal pairs sampled from the same trajectory in the dataset as "positives", and "negatives" sampled from *different* trajectories, to obtain training data pairs. The distance head is trained on both positives and negatives, whereas the action head is only trained on positives. We train the two heads jointly with supervised learning using an ℓ_2 regression loss. We use multi-GPU training with batch sizes between 400–1200 and perform gradient updates using the Adam optimizer [133] with a learning rate of 5×10^{-4}.

DEPLOYMENT: We combine this goal-reaching policy with a topological map $\mathcal{M}$, where nodes are represented by the robot's observations (augmented with the embodiment context), and edges are computed using the temporal distance estimates d from the trained policy, following the setup of ViNG [241]. At every time step, the robot associates its current and goal observations in $\mathcal{M}$, i.e., finds the node with smallest temporal distance to it, and computes the optimal sequence of subgoals $\{s_i\}$ using Dijkstra's algorithm. The policy π is queried with the current observation $\{o_t, \mathcal{C}_t\}$ and immediate subgoal s_1 to obtain a sequence of waypoints $\{\tilde{p}_i, \psi_i\}_{i=1}^{\tau}$, which are tracked by a robot-specific low-level controller.

6.5 DEPLOYING THE GNM ACROSS ROBOTS

We deploy our learned GNM omnipolicy in a variety of challenging indoor and outdoor environments on four different robot platforms. We designed out experiments to answer the following questions:

Q1. Can multi-robot training enable generalization to *novel* robots and environments?

Q2. Do GNM policies outperform policies trained solely on single-domain data?

Q3. How important are the design choices made in Sec. 6.4 for attaining good performance with the GNM?

Q4. Are policies trained with multiple datasets more robust to degradation than single-domain policies?

6.5.1 *Meet the Robots*

We deploy the GNM on four distinct robotic platforms, including a quadrotor and two other *novel* robots with no corresponding training data, as shown in Fig 31.

VIZBOT: A custom-built robot platform inspired by the design of Niwa et. al. [192], based on a Roomba. It is equipped with an off-the-shelf PCB-mounted fisheye camera. *There is no training data from a Vizbot or any other Roomba-like robot.*

DJI TELLO: A commercially available quadrotor equipped with a forward-facing camera. *There is no training data from any quadrotor for GNM.* We restrict the drone to a horizontal plane 1m off the ground, to mimic ground navigation.

CLEARPATH JACKAL UGV: A commercially available off-road platform equipped with an off-the-shelf PCB-mounted fisheye camera. *This system resembles the data collection platform used for the RECON, Berkeley, and SCAND-J datasets*, but has a different camera and mounting height.

LOCOBOT: A popular open-source platform based on a Kobuki, equipped with an off-the-shelf PCB-mounted fisheye camera. *There is no training data from a LoCoBot*, although GS was collected on a similar TurtleBot2, albeit with a different spherical camera at a lower height.

6.5.2 *Zero-Shot Deployment*

Towards answering **Q1**, we deploy the *same* trained GNM on four distinct robotic platforms *without* any fine-tuning per robot. Fig. 31 and Table 10 summarize our evaluation in a variety of indoor and outdoor environments on 4 different robots, all using the same model. Most notably, the GNM can control a Tello, despite never having seen any trajectories from aerial robots in GNM. A GNM policy consistently outperforms single robot policies across all tested robots, performing up to 5x better in some cases. We also observe generalization to massively out-of-distribution (OOD) settings, like a LoCoBot navigating outdoors on a sidewalk, or a Jackal navigating inside an office building, which were not present in the training data. This suggests that training on heterogeneous datasets can enable generalization to novel environment-robot pairs, as well as entirely new robots.

To better understand how data sharing benefits performance (**Q2**), we quantitatively evaluate the navigation performance of policies trained with heterogeneous datasets in an assortment of 20 indoor and outdoor environments on the Jackal and LoCoBot platforms (Tables 11, 12). To project the performance trends with varying amounts of data, we train policies from increasingly diverse subsets the training data — “Small”, “Mid”, and “Large”, corresponding to data from the first 2, 4, and 6 datasets listed in Table 9. We quantify performance using success rates, measured as the mean progress made towards the goal. For videos of our experiments and more information on the testing environments, please check out the supplementary video and project page.

Deploying on a LoCoBot, which is an *unseen robot* with no corresponding data present in the dataset, we find that policies trained on a single dataset (e.g., GoStanford (GS) [96] or CoryHall [123]) fail to generalize to a new embodiment with different sensors. Fine-tuning visual representations trained for task-agnostic datasets like ImageNet, which is a popular strategy for pre-training in many vision-based applications [295, 210], improves

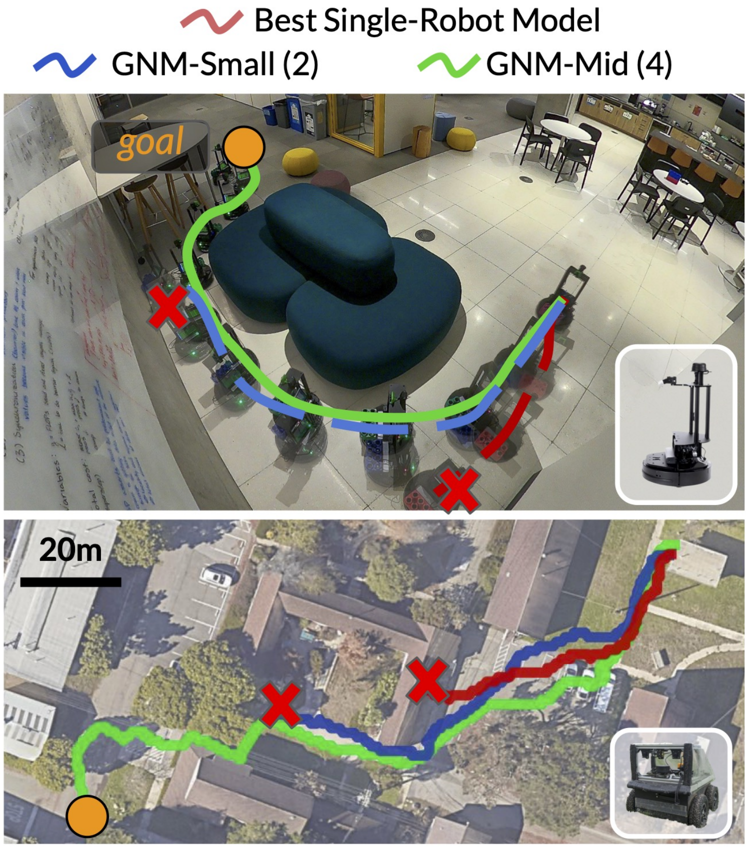


Figure 32: Qualitative comparison. Policies trained with increasingly diverse data demonstrated on a LoCoBot (top) and Jackal (bottom). Both robots were controlled by the *same* policy.

a bit but still struggles in a majority of the environments. However, policies trained by sharing task-relevant datasets across robots significantly outperform these single-domain policies, as shown in Table 11. We also observe that adding more and diverse datasets (GNM-Large) contributes towards improvements in performance, despite the additional data coming from seemingly unrelated tasks (e.g., off-road driving). Fig. 32 shows an example office environment where increasing the diversity of training data improves performance.

We observe similar trends on a Jackal, which is deployed on a variety of *previously unseen* outdoor and indoor environments (Table 12). Unsurprisingly, a single-domain policy trained on off-road RECON data [238] performs well for many outdoor environments, but struggles with navigating indoors, which is OOD for the RECON dataset. Similarly, a GS policy struggles in outdoor environments but succeeds in some easy indoor environments. GNM omnipolicies are able to generalize better to a variety of indoor and "Hard" outdoor environments, which can be over 100m long, significantly outperforming the single-domain policies (Fig. 32).

Dataset(s)	LoCoBot	Tello	Vizbot	Jackal
GS	0.26	0.21	0.51	0.31
RECON	0.62	0.79	0.26	0.68
Ours	**0.96**	**0.99**	**0.93**	**0.94**

Table 10: Summary of navigation across robots. A single policy trained on GNM-Mid outperforms the best single-robot policy for *each* robot used in our experiments, mean success rate reported.

Dataset(s)	#	Indoor		Outdoor
		Easy	Mid	
CoryHall	1	0.22	0.13	0.29
GS	1	0.25	0.16	0.44
–”– +ImageNet	1	0.35	0.35	0.57
GNM-Small	2	0.82	0.59	**1.0**
GNM-Mid	4	**1.0**	**0.97**	0.83
GNM-Large	6	**1.0**	**1.0**	0.83

Table 11: Navigation success rates on a LoCoBot. GNM omnipolicies (green) result in increasingly capable navigation, in both indoor and outdoor enviroments, on an *unseen* robot.

Dataset(s)	#	Outdoor		Indoor
		Easy	Hard	
GS	1	0.25	0.05	0.40
RECON	1	0.67	0.48	0.36
–”– +ImageNet	1	0.72	0.52	0.31
GNM-Small	2	0.75	0.52	0.42
GNM-Mid	4	**1.0**	**1.0**	**0.82**
GNM-Large	6	**1.0**	**1.0**	**0.88**

Table 12: Navigation success rates on a Jackal. By leveraging heterogeneous datasets, GNM omnipolicies (green) can drive a Jackal *better* than a policy trained on a Jackal-specific dataset (RECON), also generalizing to novel indoor environments.

Action Space	Easy	Mid
Velocities	0.73	0.54
Waypoints	0.42	0.26
Norm. Waypt.	**1.0**	**0.95**

Architecture	Easy	Mid
Stacked	0.52	0.72
Siamese	0.73	0.26
Conditioned	**1.0**	**0.95**

Context	Easy	Mid	Hard
None	**1.0**	0.79	0.36
Static	**1.0**	0.86	0.5
Temporal	**1.0**	**0.92**	**0.7**

Table 13: A systematic analysis of the design choices in Sec. 6.5.3 reveals that choosing the right action representation (left), goal-conditioned architecture (center), and conditioning on embodiment context (right) are really important to facilitate multi-robot learning.

6.5.3 *A Systematic Analysis of the Design Space*

Towards answering **Q3**, we perform a systematic analysis of the design choices presented in Sec. 6.4. We evaluate each design choice on a LoCoBot, which is an *unseen robot* with no corresponding training data, in indoor environments with varying levels of complexity, where "Easy" environments have wide passages and smooth turns, "Mid" environments have tight passages or sharp turns, and "Hard" environments are larger (up to 50m) with a combination of tight passages and multiple turns.

Shared Action Space

We compare the three action spaces discussed in Sec. 6.4.1 by training three different policies on GNM-Mid and evaluating them in 10 environments (Table 13). While using velocities as an action space works well for most easy environments, often outperforming the policy using waypoints, both these policies struggle in environments requiring dynamic maneuvers like sharp turns. A policy based on normalized waypoints, on the other hand, significantly outperforms the others, including in the challenging environments. This suggests that normalizing the action space indeed allows the policies to learn more effectively and generalize to new robots.

Embodiment Context

We consider two ways to represent the embodiment context: (i) temporally consistent context containing k consecutive past observations $\{o_{(t-k):(t-1)}\}$, and (ii) static context, containing a *fixed* set of k past observations from the robot in the target environment. Comparing these choices in environments of varying complexities (Table 13), we find that adding either form of context significantly boosts the navigation performance in harder environments, which require the robot to navigate tight passages with multiple obstacles and sharp turns. This suggests that the context helps the polices generalize better due to the additional information about the embodiment (e.g., viewpoint, speed etc.). Between the two, we found the temporal variant superior, suggesting that the temporal information (e.g., speed) is relevant to enable this generalization. In all experiments in Sec. 6.5.2 and Fig. 31, we use a temporally consistent context with $k = 5$.

Policy Architecture

We also compared different policy architectures to encode the goal information: (i) single-encoder stacking, where the observation and goal images are stacked along the channel dimension [250], (ii) a Siamese architecture, where the images are processed with independent encoders and the resulting embeddings are combined [96, 241, 289], and (iii) the conditional architecture in Fig. 30, with an additional pathway from the observation to the policy outputs [238, 185]. We found that the choice of architecture significantly affects the navigation performance, with the conditional model being the most performant. We hypothesize that this is due to the additional pathway that allows the learned embeddings to be conditioned on the current observations, leading to more generalizable representations, as studied in prior work [238].

6.5.4 *Robustness to Degradation*

A key strength of training on heterogeneous datasets is that learning across varied parameters encourages the policy to learn shared affordances across robots, thus being robust to small variation in robot parameters, such as sensor placement and mechanical properties. We show that the shared GNM can indeed offer such robustness by testing it under some example degradation scenarios shown in Fig. 33.

When testing the trained policy with a steering degradation (Fig. 33a), where the robot's maximum angular velocity is clipped, we find that the GNM can compensate for the degradation by taking a longer, smoother path towards the goal without any localization failures. We also tested the GNM while perturbing the position of the camera and physically affecting the dynamics by damaging the robot *during navigation*, and find that it can successfully reach the goals despite the degradation (Fig. 33d). Please see the supplemental video for these experiments.

6.6 DISCUSSION

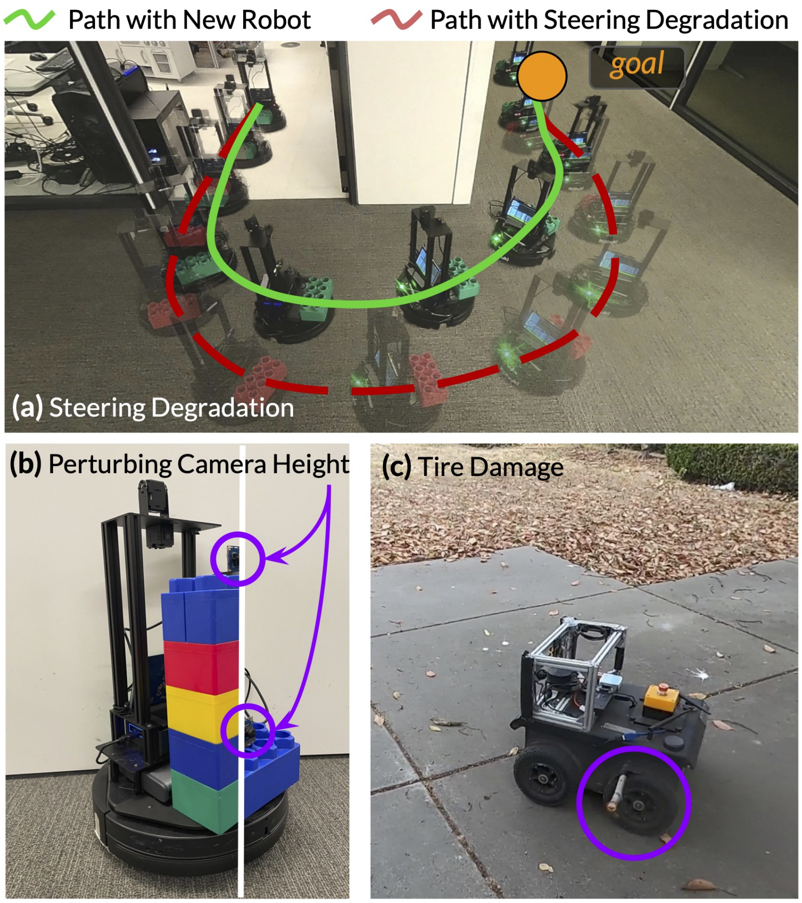
In this paper, we demonstrated that a general goal-conditioned navigation policy trained from navigation datasets collected by multiple distinct robots, ranging from RC cars to ATVs, can control *new* robots in challenging environments. The design of our learning framework is simple, and largely follows prior work: the novel observation is that a set of relatively simple decisions, such as including a temporal context and standardizing the action space, is sufficient to enable broad generalization from heterogeneous data. Empirically, we show that our approach can enable real-world navigation for a range of robots, including some not seen in training, and even an underactuated quadrotor.

Our specific instantiation of this principle does have some limitations. Most prominently, our system does not explicitly account for differences in capabilities: we assume all robots are ground robots (though we study generalization to a quadrotor) with a forward-facing RGB camera. Handling diverse sensing, actuation (beyond variability in speed and steering), and traversability, would be an exciting direction for future work. Secondly, our dataset could be much larger: while we observe exciting generalization from 60 hours of data, a much larger and broader dataset could enable even better generalization in the future.

The promise of such a *general* navigation model trained on diverse data is that it may provide a pre-trained base model for a variety of downstream navigation applications. In the same way that computer vision researchers and practitioners typically start off by downloading a pre-trained backbone to use for their task, we hope that future navigation projects might use a pre-trained navigational omnipolicy that generalizes broadly enough to offer a "universal" starting point.

ACKNOWLEDGMENTS

This research was supported by the DARPA RACER program, ARO W911NF-21-1-0097, ARL DCIST CRA W911NF-17-2-0181, AFOSR FA9550-22-1-0273, Toyota Motor North America, and Toyota Research Institute. The authors would like to thank Haresh Karnan, Xuesu Xiao, Gregory Kahn, Xiangyun Meng, and Byron Boots, for their help in aggregating the heterogeneous dataset used for training the GNM. The authors would also like to thank Brian Ichter, Antonio Loquercio, Jie Tan, Devendra Singh Chaplot, Tingnan Zhang, Laura Smith, Nick Rhinehart, Frederik Ebert, and Kelvin Xu, for useful discussions and feedback on an earlier draft of the paper.



(d)	Steering	Viewpoint	Physical
Single-Domain Policy	0.30	0.17	0.81
GNM Policy	**0.89**	**0.81**	**1.0**

Figure 33: Policies trained with GNM are more robust to degradation in parameters such as (a) actuation, (b) perturbed viewpoint, and (c) physical damage, than single-domain policies (d).

7

A FOUNDATION MODEL FOR VISUAL NAVIGATION

Synopsis

General-purpose pre-trained models ("foundation models") have enabled practitioners to produce generalizable solutions for individual machine learning problems with datasets that are significantly smaller than those required for learning from scratch. Such models are typically trained on large and diverse datasets with weak supervision, consuming much more training data than is available for any individual downstream application. In this chapter, we describe the Visual Navigation Transformer (ViNT), a *robot foundation model* that aims to bring the success of general-purpose pre-trained models to vision-based robotic navigation. ViNT is trained with a general goal-reaching objective that can be used with any navigation dataset, and employs a flexible Transformer-based architecture to learn navigational affordances and enable efficient adaptation to a variety of downstream navigational tasks. ViNT is trained on a number of existing navigation datasets, comprising hundreds of hours of robotic navigation from a variety of different robotic platforms, and exhibits *positive transfer*, outperforming specialist models trained on narrower datasets. ViNT can be augmented with diffusion-based goal proposals to explore novel environments, and can solve kilometer-scale navigation problems when equipped with long-range heuristics. ViNT can also be adapted to novel task specifications with a technique inspired by prompt-tuning, where the goal encoder is replaced by an encoding of another task modality (e.g., GPS waypoints or turn-by-turn directions) embedded into the same space of goal tokens. This flexibility and ability to accommodate a variety of downstream problem domains establish ViNT as an effective foundation model for mobile robotics.

Shah†*, Sridhar*†*, Dashora*†*, Stachowicz, Black, Hirose, Levine, "ViNT: A Foundation Model for Visual Navigation", in Annual Conference on Robot Learning (CoRL) 2023*
Project website: general-navigation-models.github.io/vint

7.1 INTRODUCTION

Recently, machine learning methods have achieved broad success in natural language processing [206], visual perception [53, 26, 85], and other domains [209, 38] by leveraging Internet-scale data to train general-purpose "foundation" models that can be adapted to new tasks by zero-shot transfer, prompt-tuning, or fine-tuning on target data [39, 239, 162, 153]. Although this paradigm has been successful in many domains, it is difficult to apply in robotics due to the sheer diversity of environments, platforms, and applications. In this paper we ask the question: *what is required of a foundation model for mobile robotics?*

In this paper, we define a *robot foundation model* as a pre-trained model that can be (i) *deployed zero-shot* in novel, useful settings (e.g., different sensors, robot embodiments, environments etc.), and (ii) *adapted* to a downstream task of choice (e.g., different objectives, goal specification types, behaviors etc.). We specifically consider the problem of visual navigation, where the robot must navigate its environment solely using egocentric visual observations. A general pre-trained robot navigation model should enable a wide range of navigation applications, readily allow fine-tuning to downstream tasks, and generalize to a broad range of environments and robotic platforms. Such a model should provide a broadly capable navigation policy on top of which applications to specific domains can be constructed, giving a base level of generalization and capabilities to new robotic platforms in zero shot that can be further improved after fine-tuning with a small amount of data.

To this end, we propose the **Vi**sual **N**avigation **T**ransformer, or ViNT: a cross-embodiment foundation model for visual navigation with strong zero-shot generalization. We train ViNT to reach goals specified by camera images, providing a very general pre-training objective that can be applied to almost any mobile robot dataset. We propose a novel exploration algorithm for the visual navigation paradigm using a diffusion model to propose short-horizon goals, and demonstrate that it enables ViNT to navigate in novel environments. ViNT can control new robots in zero-shot, explore previously unseen environments, perform indoor mapping, and navigate kilometer-scale outdoor environments without interventions. Furthermore, we show that ViNT can be fine-tuned on a small amount of data to achieve high performance with new task specification modalities – such as GPS waypoints or high-level routing commands – allowing ViNT to serve as a foundation for a wide variety of navigation applications. Lastly, we qualitatively analyze some emergent behaviors exhibited by ViNT, such as implicit preferences and navigation around dynamic pedestrians.

We hope that ViNT represents a step towards such general-purpose *robot foundation models* that can be deployed on a wide range of robots, and on a wide range of tasks, and serve as a foundation for diverse mobile robotic applications. Model weights for ViNT as well as training and deployment code are available on our project page.

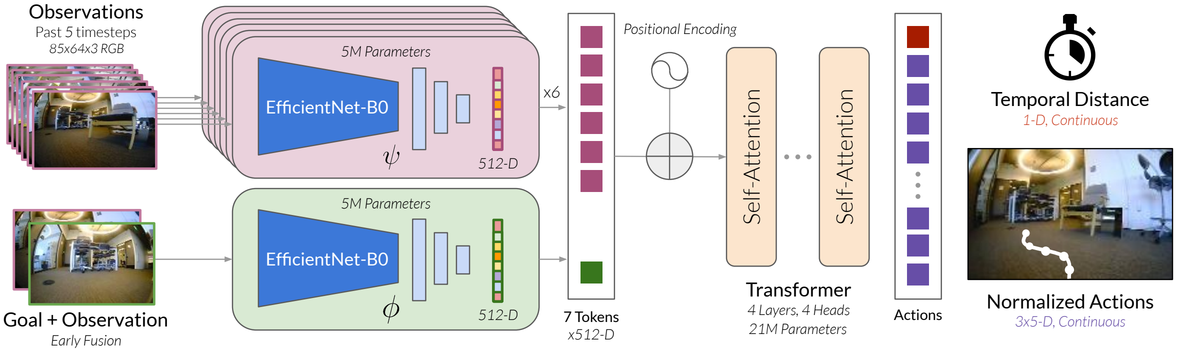


Figure 34: ViNT Model Architecture. ViNT uses two EfficientNet encoders ψ, ϕ to generate input tokens to a Transformer decoder. The resulting sequence is concatenated and passed through a fully-connected network to predict (temporal) distance to the goal as well as a sequence of $H = 5$ future actions.

7.2 RELATED WORK

Our goal is to train an effective visual navigation policy that can solve a range of downstream tasks, such as navigating to GPS goals [171], goal images [302], and skill-conditioned driving [45].Following earlier chapters, we use a combination of topological graphs for maintaining a spatial representation of the environment and learned policies for low-level control (Chapters 2, 6), and use learned heuristics to guide the robot in novel environments (Chapter 4). But unlike these works, our goal is to train a single generalist model rather than specialist solutions to each of these problems, showing how a single high-capacity model can be adapted for diverse tasks.

The closest related works to ViNT are RT-1, I2O, and GNM [270, 234, 21], which study broad generalization across environments and embodiments for robots deployed in real-world settings. While RT-1 demonstrates impressive performance in following diverse instructions, our focus is on adapting a single model across *many* robots to solve *different tasks*, by fine-tuning with small amounts of data. I2O and related efforts [270, 121] show impressive transfer from simulation to real-world environments, but we emphasize that our aim is orthogonal to the specific choice of algorithm: we focus on learning a capable navigation policy that can be efficiently adapted to solve different downstream tasks. GNM (Chapter 6) demonstrates policy learning from heterogeneous RGB datasets, but focuses on the singular task of reaching image goals in the zero-shot setting. Instead, ViNT trains a single generalist policy with an emphasis on adaptation to new embodiments and tasks in downstream applications, though it can also be used zero-shot to great effect (Sec. 7.6.1).

7.3 THE VINT MODEL

Our model is trained for image-goal navigation, providing general navigational capabilities that can then either be utilized directly, or serve as a pre-trained foundation for downstream fine-tuning with other task specifications. In the image-goal navigation task, the robot is tasked with navigating to a subgoal specified by an image observation s (i.e., the robot's observation at the goal). Unlike alternative mechanisms for goal specification such as PointGoal [7], GPS navigation, or semantic objectives [279], a model can be trained for image-goal navigation with minimal assumptions, utilizing any data that contains videos and actions, without requirements on ground-truth localization, semantic labels, or other metadata. This makes it practical to train on a large and diverse dataset sourced from many different robots, facilitating broad generalization.

ViNT takes as input current and past visual observations $o_{t-P:t}$ and a subgoal image o_s, and predicts (i) the number of time steps needed to reach the subgoal (the *dynamical* distance), and (ii) a sequence with length H of future actions leading towards the subgoal. Our 31M-parameter model, ViNT, is built on the Transformer architecture [275] and is optimized for: (i) fast and efficient inference on resource-constrained robots, and (ii) the ability to prompt and fine-tune for downstream tasks. We initialize all networks from scratch and train them end-to-end with the training objective in Eqn. 6. The model architecture is summarized in Figure 34, and described in detail in Appendix D.1.

TOKENIZATION: The ViNT architecture (Fig. 34) first tokenizes its inputs into an embedding of size $d_{\text{model}} = 512$. ViNT independently tokenizes current and $P = 5$ past visual observations by encoding them with an EfficientNet-B0 [261] model, which takes $85 \times 64 \times 3$ images as input and outputs a flattened feature vector $\psi(o_i)$ from the final convolutional layer [21].

GOAL FUSION: We found that naïvely extracting features from the goal image $\phi(o_s)$ using an EfficientNet encoder ϕ led to poor performance, often ignoring the goal entirely (see Appendix D.1). We hypothesize that effective features for image-based goal-reaching tasks are often *relative*, encoding the *difference* between the current observation and the goal rather than an absolute representation of the goal itself. Hence, we use a separate *goal fusion* encoder $\phi(o_t, o_s)$ to jointly encode the current and goal observations. We stack the two images along their channel dimensions, pass them through a second EfficientNet-B0 encoder, and flatten to obtain the goal token.

TRANSFORMER: The $P + 2$ observation and goal tokens are combined with a positional encoding and fed into a Transformer backbone f. We use a decoder-only Transformer with $n_L = 4$ multi-headed attention blocks, each with $n_H = 4$ heads and $d_{\text{FF}} = 2048$ hidden units.

TRAINING OBJECTIVE: During training, we first sample a minibatch of trajectories τ from the dataset $\mathcal{D}$. We then choose P consecutive observations to form the temporal context $o_{t:t-P}$ and randomly select a future observation $o_s := o_{t+d}$, with d sampled uniformly from $[l_{\min}, l_{\max}]$, to serve as the subgoal [91]. The corresponding H future actions $\hat{a} := a_{t:t+H}$ and the distance d are used as labels and trained with a maximum likelihood objective:

$$\mathcal{L}_{\text{ViNT}}(\phi, \psi, f) = \mathbb{E}_\tau \mathbb{E}_t \mathbb{E}_d \; [\log p(\hat{a} | f(\psi(o)_{t:t-P}, \phi(o_t, o_s)) + \lambda \log p(d | f(\psi(o)_{t:t-P}, \phi(o_t, o_s))] \tag{6}$$

where ϕ, ψ, f are as defined above, and λ balances the two losses.

EMBODIMENT-AGNOSTIC ACTION SPACE: To effectively train a single model across robots of varying sizes, speeds, and dynamics, we follow [234] and choose an embodiment-agnostic action space for ViNT. To abstract away low-level control, ViNT uses relative waypoints as its action space $\hat{a}$; to account for the large variance in speeds and sizes of the robots, we normalize these waypoints by scaling them according to the robot's top speed. During deployment, a robot-specifc controller is used to un-normalize and *track* these waypoints using low-level control.

TRAINING DATA: We train ViNT using a large-scale dataset of heterogeneous navigation trajectories from a diverse set of environments and robotic platforms with varying dynamics, camera parameters, and behaviors. The training dataset contains over 100 hours of real-world trajectories sourced entirely from existing datasets, spanning 8 distinct robotic platforms with varying speeds and dynamics. For more details about the dataset, see Appendix D.3.

DEPLOYMENT: ViNT can run on any robot equipped with an onboard camera and a low-level velocity tracking controller. Given a subgoal image s at time t, we run the model at 4Hz, and use a PD controller to track the predicted waypoints $\hat{a}$ in a receding-horizon fashion.

7.4 LONG-HORIZON NAVIGATION WITH VINT

While the goal-conditioned policy learned by ViNT captures a general understanding of navigational affordances and obstacles, it has limited applicability on its own. Many practical tasks are either not defined by goal images, or require a much longer horizon than what ViNT directly supports. We apply ViNT to several downstream applications by combining it with an episodic memory formed by a topological graph, which provides short-horizon subgoals for reaching faraway locations. In previously unseen environments, we can further augment this graph-based planner with exploratory subgoal proposals, which can drive ViNT to explore a new environment and discover a path to the goal. We consider multiple such proposal mechanisms and find that maximum

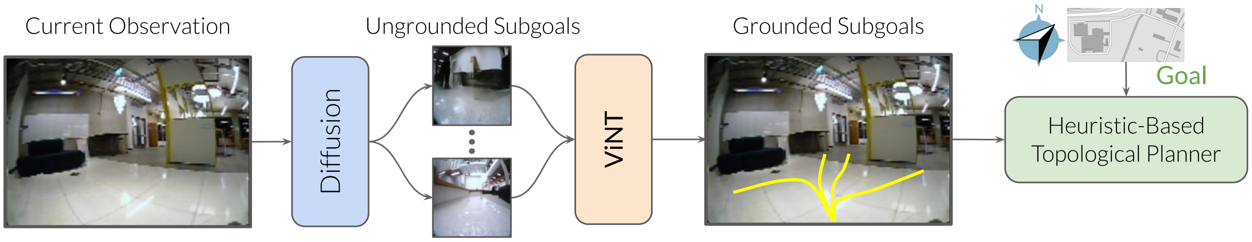


Figure 35: Long-horizon navigation in unseen environments with ViNT. We use physical search with a topological graph-based planner to explore the environment. An image-to-image diffusion model proposes diverse exploration targets which are spatially grounded using ViNT (yellow), and scored using a goal-directed heuristic h. Subgoals are added to the topological graph $\mathcal{M}$ and executed using the ViNT policy.

performance is attained by an *image diffusion model* that samples diverse future subgoal candidates conditioned on the current observation.

These subgoals are scored with a goal-directed heuristic to identify the best subgoal that makes progress towards the goal using a process akin to *physical A^* search* [232]. Past observations in the environment and unexplored frontiers are stored as nodes in a topological graph, with their connectivity determined by the distances predicted by ViNT. During exploration, we build this topological graph on the fly as the robot explores the environment. During later deployments it may be used for discovering shortcuts to arbitrary goals in the environment. We first describe the high-level algorithm that plans on top of subgoal candidates, and then discuss the process for obtaining these subgoal candidates.

7.4.1 *High-Level Planning and Exploration*

Let's assume that we have access to subgoal candidates $o_{s_i} \in \mathcal{S}$ available to ViNT for planning. We incorporate these subgoal candidates into an exploration framework for goal-directed exploration in novel environments, where the user provides a high-level goal G, which may be arbitrarily far away. We largely follow prior work [232], but swap out the learned models with ViNT and the diffusion model. We summarize the system here, and provide a more complete discussion in Appendix D.2.3.

We construct a topological graph $\mathcal{M}$ online to act as episodic memory, with each node as an individual subgoal observation and edges representing paths between two subgoals, added when the path is taken by the robot, or the model predicts a subgoal to be *reachable* from another node. We frame goal-directed exploration as a search problem, where the robot incrementally builds $\mathcal{M}$ while searching for the goal. To guide search towards the goal, the robot uses a goal-directed heuristic $h(o_t, o_{s_i}, G, \mathcal{M}, C)$ to *score* subgoal candidates according to their likelihood of reaching the goal, given additional context C — for example, a floor plan or satellite imagery [232, 270]. This heuristic may be geometric (e.g., Euclidean distance) or learned (see Appendix D.2.3).

During deployment in a new environment, the robot uses the diffusion model to generate subgoal candidates $\mathcal{S}$ from o_t, spatially grounds them using ViNT, and scores them using the goal-directed heuristic $h(.)$. The robot then selects the best subgoal o_{s^*} according to this heuristic using an A*-like planner, adds it to $\mathcal{M}$, and drives towards it using ViNT (Figure 35). During subsequent deployments in the same environment, the robot can use $\mathcal{M}$ to discover shortcuts to arbitrary goals in the environment. Please see Appendix D.2.3 for more details about the planner and heuristics. In our experiments, we consider two candidate search heuristics: a geometric heuristic based on positions of the robot and the goal, and a learned heuristic based on additional context in the form of a satellite image.

7.4.2 *Subgoal Generation with Diffusion*

The physical search algorithm presented above relies on the ability to propose subgoal candidates $\mathcal{S}$ that are both diverse and reachable from the current observation of the robot o_t. This amounts to sampling from a high-dimensional, multimodal distribution of RGB images.[1]

To do so, we train a conditional generative model $g(o_{s_i}|o_t)$ on the ViNT training data. Specifically, we apply an image-to-image diffusion model [99, 220], a generative model class that is well-suited for producing diverse samples over high-dimensional spaces such as RGB images. We train the model using randomly-sampled future observations from trajectories in the ViNT dataset (Appendix D.2.2), and sample K subgoal candidates $\mathcal{S} = \{s_1, \ldots, s_K\}$ from the model at inference time.

However, these subgoal generations are not *spatially grounded*: they do not include an actionable relationship to o_t. We ground these candidates by using ViNT to compute temporal distances $d(s_i, o_t)$ and action rollouts $a(s_i, o_t)$, yielding a set of grounded subgoals as in Fig. 38. While the samples generated by the diffusion model do not necessarily match any real observation (see Figure 35), they preserve sufficient relative features from o_t to be plausible, and we find that ViNT generalizes well to generated subgoals. We further study the behavior of this diffusion model in Section 7.6.5.

7.5 ViNT: A Foundation Model for Downstream Tasks

[1] Prior work has also studied sampling subgoals in a *latent* space [238], which may require simpler density models to learn the latent distribution. However, directly implementing this idea in our framework leads to optimization challenges and poor performance (see Section 7.6.1). Learning a sampling distribution directly in the latent space [214] is an exciting future direction, but orthogonal to the contributions of this work.

Beyond its core functionality as an image goal-conditioned model, we show that the strong navigational priors learned by ViNT can be adapted to a variety of downstream tasks, beyond navigating to image goals, by fine-tuning part or all of the model in novel environments or with new modalities of data.

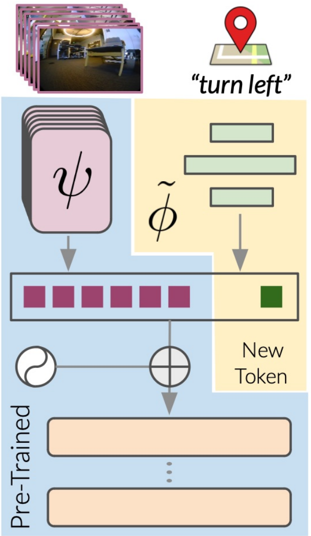


Figure 36: Adapting ViNT to different goals using a new tunable goal token.

FULL MODEL FINE-TUNING: While ViNT demonstrates strong zero-shot generalization to new environments and robots, we can further improve on-task performance by fine-tuning the entire model with the same objective but using on-task data. This allows ViNT to quickly learn new skills, forming a continually improving model. ViNT can master new environments and embodiments with as little as 1 hour of navigation data, transferring the capabilities of the original model to a new setting without retraining from scratch.

ADAPTING TO NEW MODALITIES: While specifying image goals gives a general pre-training objective, ViNT can easily be *adapted* to other common forms of goal-specification by learning a "soft prompt" mapping from the desired goal modality to the ViNT goal token [153]. We build on the Transformer architecture's ability to attend to multimodal inputs projected into a shared token space [61, 120]. Given a subgoal σ in a new modality (such as 2D coordinates or high-level routing directions [45]), we train a small neural network $\tilde{\phi}$ that maps the subgoal to this shared token space as shown in Figure 36, and replace $\phi(o_t, o_s)$. We fine-tune ViNT with on-task data $\mathcal{D}_F$ using the modified objective:

$$\mathcal{L}_{\text{adapt}} = \mathbb{E}_{\tau \in \mathcal{D}_F} \mathbb{E}_t \mathbb{E}_d \; [\log p(\hat{a} | f(\psi(o)_{t:t-P}, \tilde{\phi}(\sigma)) + \lambda \log p(d | f(\psi(o)_{t:t-P}, \tilde{\phi}(\sigma)))] \tag{7}$$

This allows adaptation to new tasks with minimal data, while still leveraging the performance and generalization of ViNT. Appendix D.2.4 includes additional details.

7.6 REAL-WORLD EVALUATION

We deployed our ViNT foundation model on five distinct robotic platforms, including a drone, a quadruped, and two other *novel* robots which are not present in the training data. We designed our experiments to answer the following questions:

Q1. Can ViNT efficiently explore previously unseen environments and incorporate heuristics?

Q2. Does ViNT generalize to *novel* robots, environments, and obstacles?

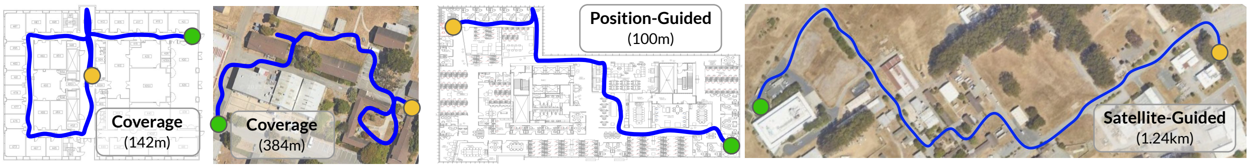


Figure 37: ViNT accomplishes long-horizon navigation with a variety of objectives in indoor and outdoor environments; example trajectories between start (orange) and goal (green) visualized here. Goal-reaching behavior can be achieved with a goal-directed heuristic (optionally guided by satellite imagery), while removing this heuristic allows for undirected exploration to maximally cover a workspace.

	Indoor	**Outdoor**
Method	Success	Success
End-to-End BC	0.72	0.44
End-to-End GCG [123]	—	0.61
RECON	0.19	0.23
ViNT-R (Random Subgoals)	0.81	—
ViNT	**0.94**	**1.00**

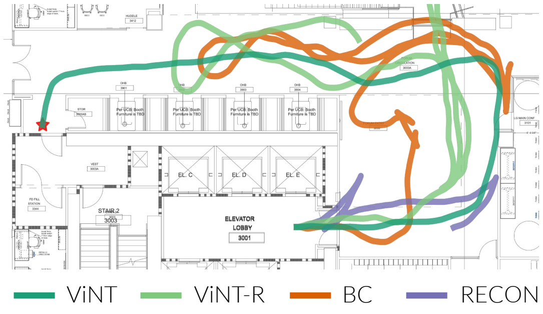


Table 14: ViNT paired with our physical search algorithm consistently outperforms baselines for the task of undirected goal-reaching in indoor and outdoor environments (*left*). By effectively planning over diffusion subgoal proposals, ViNT is able to find an efficient path to the goal. Other baselines struggle to explore large indoor environments, shown by trajectories overlaid on an indoor floor plan (*right*).

Q3. Can ViNT be fine-tuned to improve performance in out-of-distribution environments?

Q4. Can the ViNT policy be adapted to handle new task specification and modalities?

Please see Appendix D.4 for more details on platforms used in the training data and in evaluation. We did not perform any careful system identification or homogenization of sensors across the robots and datasets; all datasets are used as obtained from their original sources, and every robot platform has its own low-level controller and onboard stack.

7.6.1 *Navigation Performance*

Towards understanding **Q1**, we deploy our full graph-based navigation pipeline (Section 7.4.1) in a variety of challenging indoor and outdoor environments, previously unseen in the training data. We evaluate the performance of ViNT on two challenging tasks: (i) coverage exploration, where the objective is maximally explore an environment

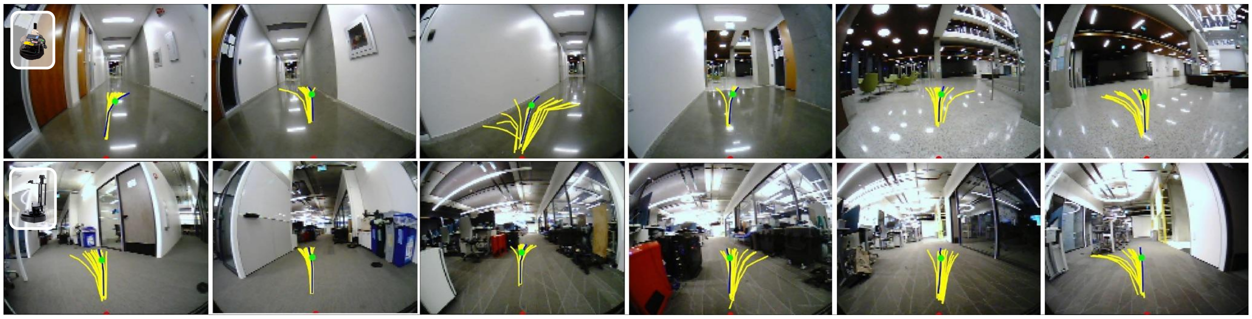


Figure 38: Visualizing ViNT exploration rollouts in challenging indoor environments using the Vizbot (top) and LoCoBot (bottom) robotic platforms. Future action samples $\hat{a}$ obtained by *spatially grounding* the subgoal candidates are shown in yellow, with the best actions corresponding to the best candidate marked in blue.

	Indoor: Position		Outdoor: GPS			Outdoor: Satellite		
Method	Success	Distance (m)	Success	SPL	Distance (m)	Success	SPL	Distance (m)
ViKiNG [232]	0.60	56	0.64	0.42	720	0.77	0.68	780
ViNT	**0.90**	**91**	**0.95**	**0.84**	**1270**	**1.00**	**0.94**	**1040**

Table 15: ViNT can effectively utilize goal-directed heuristics, such as 2D goal positions and satellite images, to explore novel kilometer-scale environments successfully and without interventions.

in search of a goal whose location is unknown, and (ii) guided exploration, where the objective is to reach a goal using contextual information such as GPS coordinates or a satellite image (see Figure 37 for task examples). We compare ViNT to a variety of baselines, including end-to-end policies trained with imitation or RL [123, 270], a prior graph-based approach using a VIB for exploration [238], and an ablation of ViNT that samples random images from the training set to use as goals rather than generating subgoals with a diffusion model. See Appendix D.5.1 for details about the experimental setup.

For the coverage exploration task, the agent is placed in an unknown environment and tasked with exploring the environment maximally in search of the goal without any additional cues. Table 14 summarizes the success rates for this task in indoor and outdoor environments. We find that, while end-to-end baselines avoid collisions with their surroundings, they fail to explore new areas and often get stuck in small areas of the environment. Graph-based methods avoid this trap by explicitly reasoning about coverage with the search objective, leading to a high success rate for ViNT. Qualitative analysis (Table 14-*right*) shows that planning over the diverse subgoals proposed using diffusion leads to more efficient paths, whereas other baselines take winding paths while exploring. Figure 37 illustrates the egocentric rollouts of the coverage exploration task in

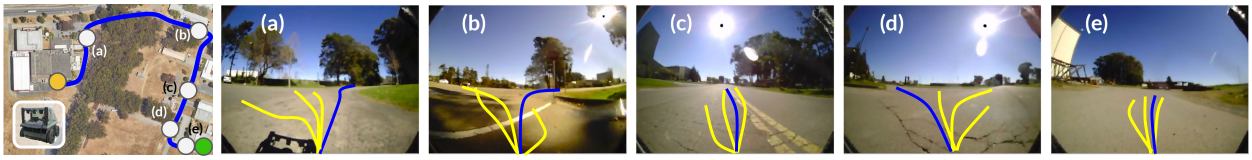


Figure 39: Satellite-guided physical search with ViNT. We visualize a 765m rollout of ViNT with a satellite image-based heuristic from start (orange) to goal (green). The future action samples $\hat{a}$ obtained by *spatially grounding* the subgoal candidates for five instances in the trajectory are shown in yellow. An A*-like planner uses the heuristic to pick the best subgoal (corresponding $\hat{a}$ marked in blue), guiding the robot to the goal.

challenging indoor environments. ViNT-R performs respectably despite the lack of valid subgoal proposals. See Section 7.6.5 for a discussion on this observation.

This observation extends to the position-guided navigation task (Table 15), where the robots are tasked with reaching a *2D goal position* in a previously unseen environment. The robots have access to onboard wheel odometry (indoors), GPS coordinates (outdoors), or passive satellite imagery (outdoors), to track their position and use as a goal-directed heuristic. Compared to a baseline of the previous state of the art [232], we find that the various sub-goal predictions from the diffusion model paired with the graph-based scoring scheme lead to a higher success rate and a greater distance traveled without collisions. ViNT is also more effective at avoiding collisions in crowded indoor spaces, and more efficient at reaching goals in outdoor spaces (captured by the SPL metric), owing to the implicit affordances and preferences learned by the large-scale pre-training (see further analysis in Section 7.6.5. ViNT also requires fewer interventions, observed by the larger distance before observed collisions. Figure 39 illustrates a rollout of physical search in an outdoor environment with ViNT using satellite image as context (also see Figure 37).

7.6.2 *Zero-Shot Generalization: a Single Policy to Drive Any Robot*

Towards answering **Q2**, we deploy the *same* pre-trained ViNT policy on four distinct robotic platforms *without* any fine-tuning for the task of undirected exploration. We report the maximum displacement of the robot (in meters) from its starting position, without interventions, as a proxy for reaching arbitrarily distant goals in complex environments in Table 16. Most notably, ViNT successfully generalizes zero-shot to control a Go 1 quadruped, which does not appear during training.

We compare ViNT trained across all the combined datasets and robots to the best single-robot baseline — a model trained using data only from the target environment — as well as the GNM model (Chapter 6) trained on

Model	LoCoBot	Go 1	Vizbot	Jackal
Single-Robot	40	12	40	184
GNM [234]	60	8	20	**427**
ViNT	**120**	**45**	**110**	**438**

Table 16: In coverage tasks, ViNT drives different robots for 100s of meters (reported maximum displacement without intervention), beating lower-capacity models (GNM)

all datasets. We observe that policies trained across robot embodiments can not only match, but also *outperform*, single-robot models across all the embodiments we studied. We also find that the larger capacity of ViNT leads to improved generalization compared to the smaller GNM model, especially on robots that do not appear in the training dataset (e.g., Go 1). Crucially, we also find that ViNT demonstrates *positive transfer* for in-domain robots (Vizbot), greatly outperforming a specialist model trained on only the target robot and setting, an emergent phenomenon not present in smaller models. This indicates that the model generalizes between tasks to improve performance, a key property of a foundation model.

7.6.3 *Broader Generalization via Fine-Tuning*

To answer **Q3**, we consider the problem of fine-tuning ViNT in the low data regime. In this setting, the entire ViNT model is fine-tuned end-to-end with a reduced learning rate of 1×10^{-4} over $n_{\text{ep}} = 5$ epochs (Section 7.5). We assume access to a small amount of on-task data (at most 5 hours, with successful results in 1-2 hours of data), and study the the efficiency of learning from subsets of this data using ViNT. We study fine-tuning for the task of autonomous driving in the CARLA simulator for two reasons: (i) the simulated CARLA environment is perceptually distinct from the real-world data used to train ViNT (Fig. 40), and (ii) the on-road driving task requires very specific semantic behavior, i.e., driving in a lane and making smooth turns, that is not present in our real-world training data. We show that ViNT can be fine-tuned on a small amount of data (under 1 hour) to achieve strong performance by effectively leveraging the navigational priors encoded in the model.

We compare the ViNT backbone to several alternatives, including visual representations trained with supervised learning [53], unsupervised objectives [39, 207, 167], and an embodiment-agnostic navigation policy [234]. We use the same fine-tuning data and procedure for all models (see Section 7.5); please see Appendix D.5.3 for more details.

Table 3 summarizes our findings. We report fractional progress towards the goal as "success", and the fraction of trajectories where the agent drives within the driving lane as "in lane". While pre-trained visual representations significantly improve task performance over a policy trained entirely from scratch, we observe that the learned

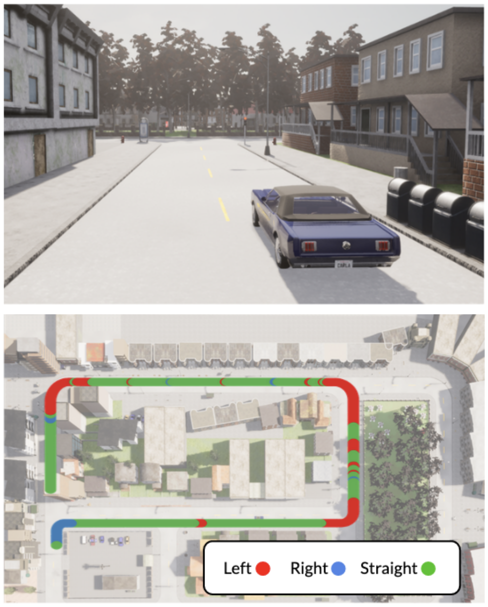


Figure 40: The CARLA test environment (*top*), and a bird's eye view show-

	Images		**Positions**	**Routing**
Method	Success	In Lane	Success	Success
Scratch	0.45	0.74	0.79	0.43
ImageNet	0.22	0.71	0.59	0.45
SimCLR [39]	0.21	0.63	0.70	0.64
VC-1 [167]	0.19	0.65	0.49	0.38
GNM [234]	0.49	0.66	0.45	0.49
ViNT	**0.82**	**0.82**	**0.89**	**0.72**

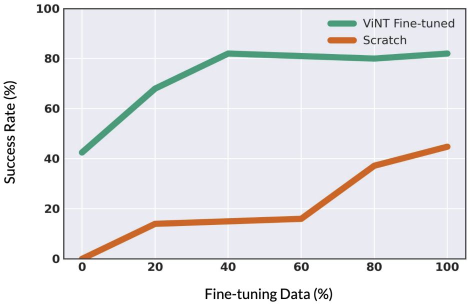


Table 3: *Left:* ViNT can be fine-tuned end-to-end (**Images**) or adapted to downstream tasks (**Positions** and **Routing**), and outperforms training from scratch and other pre-training methods. *Right:* ViNT can transfer navigational affordances to novel tasks (40% success without fine-tuning), and efficiently masters the task (80% success) with less than 1 hour of fine-tuning data. ViNT fine-tuning (green) outperforms a single-domain model trained with 5× data (orange).

policies suffer from frequent collisions and poor performance. GNM [234] outperforms these baselines due to its strong navigation prior, but the lower-capacity model is unable to generalize fully to the task. ViNT, on the other hand, is able to achieve strong performance, achieving substantially higher success rate than the next best baseline. Sweeping over fine-tuning dataset size (Table 3-right) shows that ViNT achieves strong performance with as little as 1 hour of fine-tuning data, demonstrating its ability to generalize to new environments with very little data.

7.6.4 *Adapting ViNT to Downstream Tasks*

To evaluate **Q4**, we investigate whether ViNT can serve as a foundation model for a broader range of downstream tasks by considering goal modalities beyond subgoal images (see Section 7.6.4). We consider the same CARLA driving task but with two different high-level planners: (i) a position-based planner that commands a sequence of GPS waypoints, and (ii) a routing planner with similar functionality to Google Maps that commands high-level navigation directions (left/right/straight) to the policy [45]. We compare the pre-trained navigational priors learned by ViNT to the baselines discussed earlier, corresponding to pre-trained visual representations and policies, each adapted to the downstream task using the same on-task data (see Appendix D.5.3 for more details).

Table 3 summarizes our results for the two tasks. We again find that general pre-trained visual representations, such as ImageNet or VC-1, are not sufficient to extract navigational affordances for challenging downstream tasks, suggesting that effective generalization requires more than general visual representations [196, 167]. We also find that unlike fine-tuning, GNM [234] struggles with the adaptation task, suggesting that the architecture and increased capacity of ViNT are essential for broad generalization and adaptation. ViNT achieves strong performance on both tasks, demonstrating its ability to serve as a foundation model for downstream tasks.

7.6.5 *Emergent Behaviors*

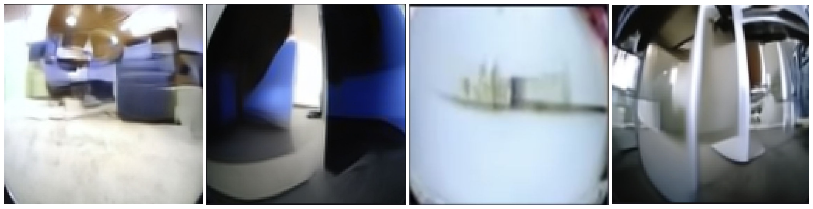


Figure 41: Samples from the diffusion model may be invalid subgoals, but ViNT is robust to such proposals.

One of the most exciting aspects of large-scale machine learning is the potential for emergent behavior that arises from the training of large models on diverse datasets. Despite the simple self-supervised training objective used by ViNT (Equation 6), it shows a number of emergent behaviors, which we describe qualitatively in this section and present as demos on the project page.

implicit navigation affordances: Ideally, we would like a robot foundation model to exhibit some desirable "default" behavior, while providing a mechanism for downstream applications to adapt this behavior as needed. We find that ViNT has this property vis-a-vis collision-avoidance. One piece of evidence is its behavior when provided with *random* subgoals from locations that are not reachable by the robot, studied quantatively via the ViNT-R baseline in Table 14. In this case, despite the subgoals being invalid and out-of-distribution (ViNT was only trained to reach subgoals), ViNT succeeds at exploring the environment and reaches the goal 80% of the time, outperforming all baselines. This suggests that ViNT takes collision-free actions when provided with meaningless goals (i.e. the above "default"), while still attempting to follow reachable subgoals.

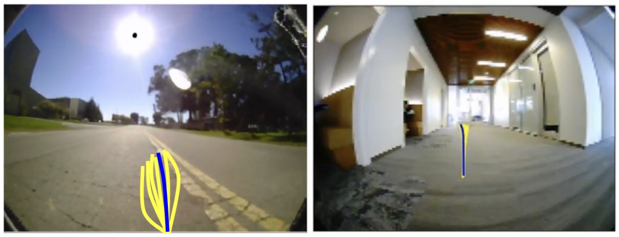


Figure 42: ViNT exhibits an implicit preference for following paved roads (*left*) and hallways (*right*).

Indeed, although the "full" version of our method augmented with the diffusion model performs better, the subgoals generated by this model are often of low quality with many artifacts, and sometimes do not match any real reachable state (Figure 41). Nonetheless, because of this "default" behavior, ViNT is able to successfully leverage the valid subgoals, while ignoring the bad ones, and demonstrate collision-free

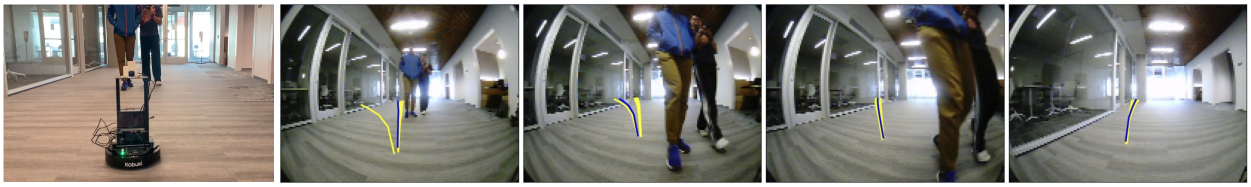


Figure 43: Robustness to dynamic pedestrians. ViNT can successfully navigate around a crowd of dynamic pedestrians and reach the goal behind them, despite its simple self-supervised training objective.

navigation in previously unseen environments.

IMPLICIT NAVIGATION PREFERENCES: Yet another interesting property exhibited by ViNT is its implicit preference to follow paved roads (outdoors), and drive smoothly in the middle of hallways (indoors), as demonstrated in Figure 42 and in the supplemental video. This is particularly interesting since a large chunk of the pre-training dataset contains suboptimal, weavy trajectories, and suggests that ViNT can learn "good" default behavior from the diverse training behaviors. This preference helps ViNT efficiently explore previously unseen environments, where other baselines tend to explore the environment haphazardly (see Table 14 (*right*)).

ROBUSTNESS TO DYNAMIC PEDESTRIANS: While ViNT is trained only on offline data with a simple, self-supervised training objective, we find that its collision avoidance capabilities generalize to dynamic obstacles and pedestrians. Figure 43 exhibits an instance where the robot is tasked with navigating to a goal behind two pedestrians. ViNT selects actions that avoid the pedestrians and recovers to the original path, successfully reaching the goal.

7.7 DISCUSSION

We presented ViNT, a robot foundation model that is trained for a generic image-goal navigation task on diverse data from many different robots, and can then support a variety of different navigation functionalities. ViNT can be deployed for long-horizon navigation in combination with a topological graph planning method, explore new environments with goals proposed by a diffusion model, be fine-tuned to new domains (such as autonomous driving), and be adapted to new task specification methods, such as GPS coordinates or turn-by-turn routing commands. Our results show that ViNT can successfully generalize across robots and environments, outperforms prior navigational models, can be efficiently fine-tuned to new domains and tasks, and shows promising emergent behaviors such as navigating through dynamic pedestrians.

ACKNOWLEDGMENTS

This research was partly supported by ARL DCIST CRA W911NF-17-2-0181, ARO W911NF-21-1-0097, IIS-2150826, and compute from Google TPU Research Cloud, NVIDIA, NSF CloudBank, and the Berkeley Research Computing program. The authors would like to thank Yunhao Cao, Seung-Hyun Kwon, Hrish Leen, Laura Smith, Medini Tolpadi, and Christopher Yu, for help with setting up experiments; Ted Xiao, for help with reproducing RT-1; Chethan Bhateja, Devendra Singh Chaplot, Kuan Fang, Adrien Gaidon, Dibya Ghosh, Saurabh Gupta, Haresh Karnan, Vijay Kumar, Hrish Leen, Fangchen Liu, Arjun Majumdar, Carlos Nieto, Aravind Rajeswaran, Ethan Stump, Jie Tan, Joanne Truong, and Tingnan Zhang, for useful discussions and feedback during various stages of this research. The authors would also like to thank Byron Boots, Gregory Kahn, Haresh Karnan, Xiangyun Meng, and Xuesu Xiao, for their help in aggregating the dataset used for training ViNT.

8

GOAL MASKED DIFFUSION POLICIES FOR UNIFIED NAVIGATION AND EXPLORATION

Synopsis

In this chapter, we propose a novel architecture for increasing the expressivity of robot foundation models so they can represent complex, multimodal action distributions in challenging environments. We describe how we can train a single unified diffusion policy to handle both goal-directed navigation and goal-agnostic exploration, with the latter providing the ability to search novel environments, and the former providing the ability to reach a user-specified goal once it has been located. We show that this unified policy results in better overall performance when navigating to visually indicated goals in novel environments, as compared to approaches that use subgoal proposals from generative models, or prior methods based on latent variable models. We instantiate our method by extending the ViNT cross-embodiment policy with a diffusion model decoder to flexibly handle both goal-conditioned and goal-agnostic navigation. Our experiments, conducted on a real-world mobile robot platform, show effective navigation in unseen environments in comparison with five alternative methods, and demonstrate significant improvements in performance and lower collision rates, despite utilizing smaller models than state-of-the-art approaches.

8.1 INTRODUCTION

Robotic learning provides us with a powerful tool for acquiring multi-task policies that, when conditioned on a goal or another task specification, can perform a wide variety of different behaviors. Such policies are appealing not only because of their flexibility, but because they can leverage data from a variety of tasks and domains and, by sharing knowledge across these settings, acquire policies that are more performant and more

Sridhar, Shah, Glossop, Levine, "NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration", in IEEE International Conference on Robotics and Automation (ICRA) 2024
Project website: general-navigation-models.github.io/nomad

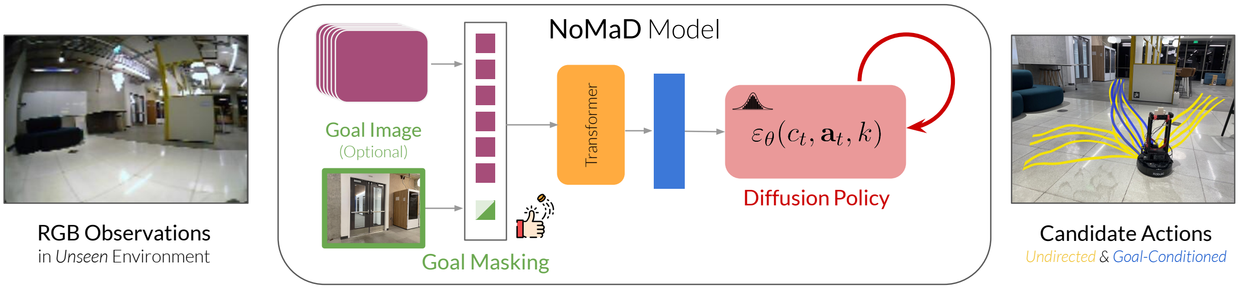


Figure 44: NoMaD is the first flexibly conditioned diffusion model of robot actions that can perform both goal-conditioned navigation and undirected exploration in previously unseen environments. It uses goal masking to condition on an optional goal image, and a diffusion policy to model complex, multimodal action distributions in challenging real-world environments.

generalizable. However, in practical settings, we might encounter situations where the robot doesn't know *which* task to perform because the environment is unfamiliar, the task requires exploration, or the direction provided by the user is incomplete. In this work, we study a particularly important instance of this problem in the domain of robotic navigation, where the user might specify a destination visually (i.e., via a picture), and the robot must locate this destination by *searching* its environment. In such settings, standard multi-task policies trained to perform the user-specified task are not enough by themselves: we also need some way for the robot to explore, potentially trying different tasks (e.g., different possible destinations for searching the environment), before figuring out how to perform the desired task (i.e., locating the object of interest). Prior works have often addressed this challenge by training a separate high-level policy or goal proposal system that generates suitable exploratory tasks, for example using high-level planning [70], hierarchical reinforcement learning [156], and generative models [242]. However, this introduces additional complexity and often necessitates task-specific mechanisms. Can we instead train a single highly expressive policy that can represent *both* task-specific and task-agnostic behavior, utilizing the task-agnostic behavior for exploration and switching to task-specific behavior as needed to solve the task?

In this paper, we present a design for such a policy by combining a Transformer backbone for encoding the high-dimensional stream of visual observations with diffusion models for modeling a sequence of future actions and instantiate it for the particular problem of visual exploration and goal-seeking in novel environments. Our main insight is that such an architecture is uniquely suited for modeling task-specific and task-agnostic pathways since it provides high capacity (both for modeling perception and control) and the ability to represent complex, multimodal distributions.

The main contribution of our work is **N**avigati**o**n with Goal **Ma**sked **D**iffusion, or NoMaD, a novel architecture for robotic navigation in previously unseen environments that uses a unified diffusion policy to jointly represent exploratory task-agnostic behavior and goal-directed task-specific behavior in a framework that combines graph search, frontier exploration, and highly expressive policies. We evaluate the performance of NoMaD on both undirected and goal-conditioned experiments across challenging indoor and outdoor environments, and report improvements over the state-of-the-art, while also being 15× more computationally efficient. To the best of our knowledge, NoMaD is the first successful instantiation of a goal-conditioned action diffusion model, as well as a unified model for both task-agnostic and task-oriented behavior, deployed on a physical robot.

8.2 RELATED WORK

The closest related work to NoMaD is ViNT, which uses a goal-conditioned navigation policy in conjunction with a *separate* high-capacity subgoal proposal model (Chapter 7). The subgoal proposal model in ViNT is instantiated as a 300M parameter image diffusion model [99], generating candidate subgoal images conditioned on the robot's current observation. NoMaD uses diffusion models differently: rather than generating subgoal images with diffusion and conditioning on these generations, we directly model actions conditioned on the robot's observation with diffusion. Empirically, we find that NoMaD outperforms the ViNT system by over 25% in undirected exploration. Furthermore, since NoMaD does not generate high-dimensional images, it requires over 15× fewer parameters, providing a more compact and efficient approach that can run directly on the less powerful onboard computers (e.g., NVIDIA Jetson Orin).

A key challenge with predicting sequences of robot actions for exploration is the difficulty in modeling multimodal action distributions. Prior work has addressed this by exploring different action representations, such as autoregressive predictions of quantized actions [180, 47, 230, 31], using latent variable models [238, 68], switching to an implicit policy representation [72], and, most recently, using conditional diffusion models for planning and control [115, 278, 42, 199, 212, 90]. State- or observation-conditioned diffusion models of action [278, 42] are particularly powerful, since they enable modeling complex action distributions without the cost and added complexity of inferring future states/observations. NoMaD extends this formulation by additionally conditioning the action distribution on both the robot's observations and the optional goal information, giving the first instantiation of a "diffusion policy" that can work in both goal-conditioned and undirected modes.

8.3 PRELIMINARIES

Our objective is to design a control policy π for visual navigation that takes the robot's current and past RGB observations as input $\mathbf{o}_t := o_{t-P:t}$ and outputs a distribution over future actions $\mathbf{a}_t := a_{t:t+H}$. The policy may additionally have access to an RGB image of a goal o_g, which can be used to specify the navigation *task*. When a goal o_g is provided, π must take actions that make progress towards the goal, and eventually reach it. In

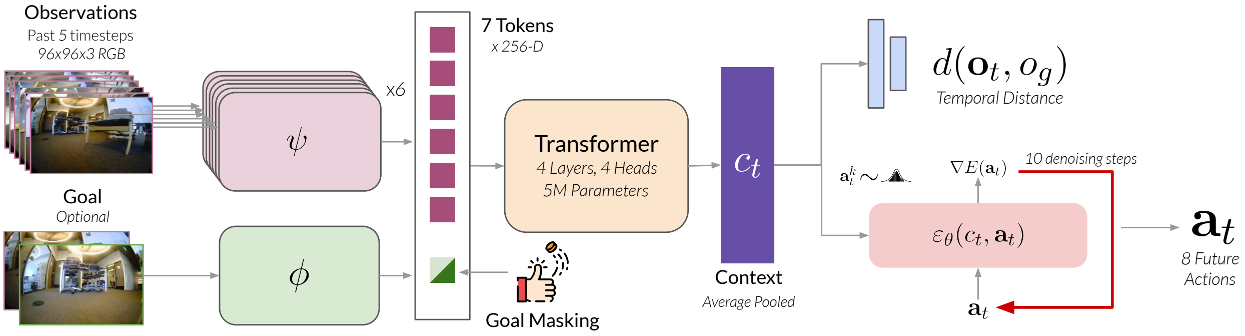


Figure 45: Model Architecture. NoMaD uses two EfficientNet encoders ψ, ϕ to generate input tokens to a Transformer decoder. We use *goal masking* to jointly reason about task-agnostic and task-oriented behaviors through the observation context c_t. We use action diffusion conditioned on the context c_t to obtain a highly expressive policy that can be used in both a goal-conditioned and undirected manner.

an unseen environment, the goal image o_g may not be available, and π must *explore* the environment by taking safe and reasonable navigation actions (e.g., avoiding obstacles, following hallways etc.), while providing sufficient coverage of the valid behaviors in the environment. To facilitate long-horizon exploration and goal-seeking, we follow the setup of ViKiNG [232] and pair $\pi(\mathbf{o}_t)$ with a topological memory of the environment $\mathcal{M}$, and a high-level planner that encourages the robot to explore the environment by navigating to unexplored regions.

VISUAL GOAL-CONDITIONED POLICIES: To train goal-conditioned policies for visual inputs, we follow a large body of prior work on training high-capacity policies based on the Transformer architecture [275, 21, 190, 242]. Specifically, we use the Visual Navigation Transformer (ViNT) [242] policy as the backbone for processing the robot's visual observations $\mathbf{o}_t$ and goal o_g. ViNT uses an EfficientNet-B0 encoder [261] $\psi(o_i)$ to process each observation image $i \in \{t-P, \ldots, t\}$ independently, as well as a goal fusion encoder $\phi(o_t, o_g)$ to tokenize the inputs. These tokens are processed using multi-headed attention layers $f(\psi(o_i), \phi(o_t, o_g))$ to obtain a sequence of context vectors that are concatenated to obtain the final context vector c_t. The context vector is then used to predict future actions $\mathbf{a}_t = f_a(c_t)$ and temporal distance between the observation and the goal $d(o_t, o_g) = f_d(c_t)$, where f_a, f_c are fully-connected layers. The policy is trained using supervised learning using a maximum likelihood objective, corresponding to regression to the ground-truth actions and temporal distance. While ViNT shows state-of-the-art performance in goal-conditioned navigation, it cannot perform undirected exploration and requires an external subgoal proposal mechanism. NoMaD extends ViNT to enable both goal-conditioned and undirected navigation.

EXPLORATION WITH TOPOLOGICAL MAPS: While goal-conditioned policies can exhibit useful affordances and collision-avoidance behavior, they may be insufficient for navigation in large environments that require reasoning over long time horizons. To facilitate long-horizon exploration and goal-seeking in large environments, we follow the setup of ViKiNG [232] and integrate the policy with episodic memory $\mathcal{M}$ in the form of a topological graph of the robot's experience in the environment. $\mathcal{M}$ is represented by a graph structure with nodes corresponding to the robot's visual observations in the environment, and edges corresponding to navigable paths between two nodes, as determined by the policy's goal-conditioned distance predictions. When navigating large environments, the robot's visual observations $\mathbf{o}_t$ may not be sufficient to plan long-horizon trajectories to the goal. Instead, the robot can use the topological map $\mathcal{M}$ to plan a sequence of subgoals that guide the robot to the goal. When exploring previously unseen environments, we construct $\mathcal{M}$ online while the robot searches the environment for a goal. Beyond undirected coverage exploration, this graph-based framework also supports the ability to reach high-level goals G, which may be arbitrarily far away and specified as GPS positions, locations on a map, language instructions, etc. In this work, we focus on frontier-based exploration, which tests the ability of NoMaD to propose diverse subgoals and search unseen environments. We largely follow the setup of prior work [232], swapping the learned policy with NoMaD.

8.4 METHOD

Unlike prior work that uses separate policies for goal-conditioned navigation and open-ended exploration [242], we posit that learning a single model for both behaviors is more efficient and generalizable. Training a shared policy across both behaviors allows the model to learn a more expressive prior over actions $\mathbf{a}_t$, which can be used for both conditional and unconditional inference. In this section, we describe our proposed NoMaD architecture, which is a goal-conditioned diffusion policy that can be used for both goal-reaching and undirected exploration. The NoMaD architecture has two key components: (i) attention-based goal-masking, which provides a flexible mechanism for conditioning the policy on (or masking out) an optional goal image o_g; and (ii) a diffusion policy, which provides an expressive prior over collision-free actions that the robot can take. Figure 45 shows an overview of the NoMaD architecture, and we describe each component in detail below.

8.4.1 *Goal Masking*

In order to train a shared policy for goal-reaching and undirected exploration, we modify the ViNT architecture described in Section 8.3 by introducing a binary "goal mask" m, such that $c_t = f(\psi(o_i), \phi(o_t, o_g), m)$. m can be used to mask out the goal token $\phi(o_t, o_g)$, thus blocking the goal-conditioned pathway of the policy. We implement masked attention by setting the goal mask $m = 1$, such that the downstream computation of c_t does not attend to the goal token. We implement unmasked attention by setting $m = 0$, such that the goal token is used alongside observation tokens in the downstream computation of c_t. During training, the goal mask m is sampled from a Bernoulli distribution with probability p_m. We use a fixed $p_m = 0.5$ during training, corresponding to equal number of training samples corresponding to goal-reaching and undirected exploration. At test time, we set m corresponding to the desired behavior: $m = 1$ for undirected exploration, and $m = 0$ for reaching user-specified goal images. We find that this simple masking strategy is very effective for training a single policy for both goal-reaching and undirected exploration.

8.4.2 *Diffusion Policy*

While goal masking allows for a convenient way to condition the policy on a goal image, the distribution over actions that results from this, particularly when a goal is *not* provided, can be very complex. For example, at a junction, the policy might need to assign high probabilities to left and right turns, but low probability to any action that might result in a collision. Training a single policy to model such complex, multimodal distributions over action sequences is challenging. To effectively model such complex distributions, we use a diffusion model [99] to approximate the conditional distribution $p(\mathbf{a}_t|c_t)$, where c_t is the observation context obtained after goal masking.

We sample a sequence of future actions $\mathbf{a}_t^K$ from a Gaussian distribution and perform K iterations of *denoising* to produce a series of intermediate action sequences with decreasing levels of noise $\{\mathbf{a}_t^K, \mathbf{a}_t^{K-1}, \ldots, \mathbf{a}_t^0\}$, until the desired noise-free output $\mathbf{a}_t^0$ is formed. The iterative denoising process follows the equation

$$\mathbf{a}_t^{k-1} = \alpha \cdot (a_t^k - \gamma\epsilon_\theta(c_t, \mathbf{a}_t^k, k) + \mathcal{N}(0, \sigma^2 I)) \tag{8}$$

where k is the number of denoising steps, ϵ_θ is a noise prediction network parameterized by θ, and α, γ and σ are functions of the noise schedule.

The noise prediction network ε_θ is conditioned on the observation context c_t, which may or may not include goal information, as determined by the mask m. Note that we model the conditional (and not joint) action distribution, excluding c_t from the output of the denoising process, which enables real-time control and end-to-end training of the diffusion process and visual encoder. During training, we train ε_θ by adding noise to ground truth action sequences. The predicted noise is compared to the actual noise through the mean squared error (MSE) loss.

8.4.3 *Training Details*

The NoMaD model architecture is illustrated in Figure 45. We train NoMaD on a combination of GNM and SACSoN datasets, large heterogeneous datasets collected

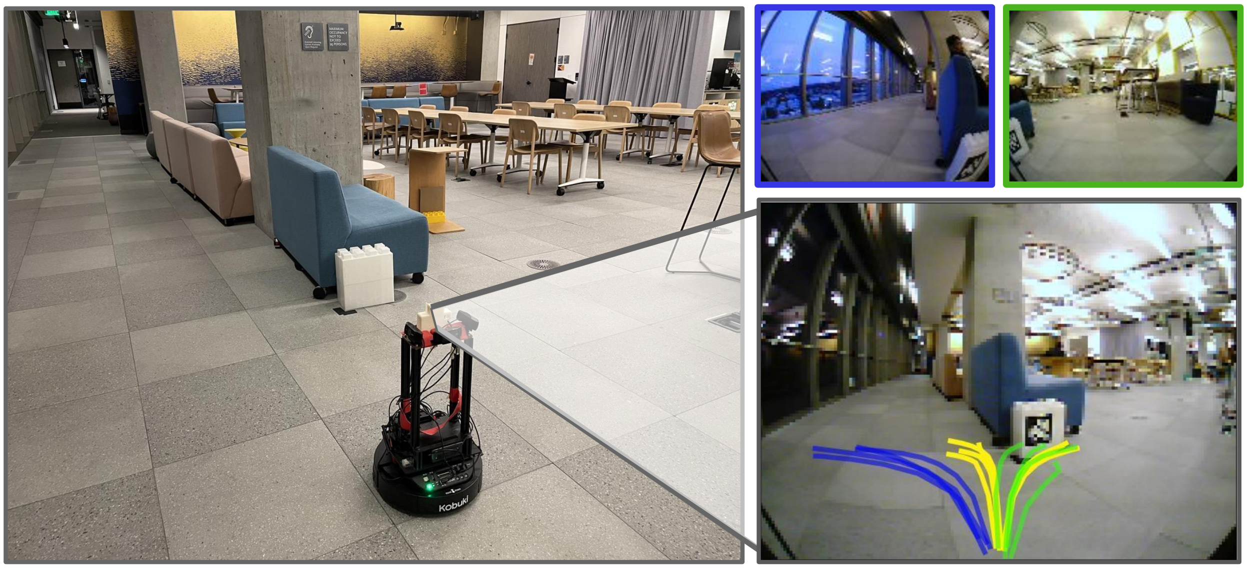


Figure 46: Visualizing the task-agnostic (yellow) and goal-directed pathways for two goal images (green, blue) learned by NoMaD. NoMaD predicts a bimodal distribution of collision-free actions in the absence of a goal, and *snaps* to a narrower distribution after conditioning on two different goal images.

across a diverse set of environments and robotic platforms, including pedestrian-rich environments, spanning over 100 hours of real-world trajectories [234, 98]. NoMaD is trained end-to-end with supervised learning using the following loss function:

$$\begin{aligned}\mathcal{L}_{\text{NoMaD}}(\phi, \psi, f, \theta, f_d) = & \, MSE(\varepsilon^k, \varepsilon_\theta(c_t, \mathbf{a}_t^0 + \varepsilon^k, k)) \\ & + \lambda \cdot MSE(d(\mathbf{o}_t, o_g), f_d(c_t))\end{aligned} \tag{9}$$

where ψ, ϕ correspond to visual encoders for the observation and goal images, f corresponds to the Transformer layers, θ corresponds to the parameters of the diffusion process, and f_d corresponds to the temporal distance predictor. $\lambda = 10^{-4}$ is a hyperparameter that controls the relative weight of the temporal distance loss. During training, we use a goal masking probability $p_m = 0.5$, corresponding to an equal number of goal-reaching and undirected exploration samples. The diffusion policy is trained with the Square Cosine Noise Scheduler [191] and $K = 10$ denoising steps. We uniformly sample a denoising iteration k, and we also sample a corresponding noise ϵ^k with the variance defined at iteration k. The noise prediction network, ϵ_θ, consists of a 1D conditional U-Net [115, 42] with 15 convolutional layers.

We use the AdamW optimizer [164] with a learning rate of 10^{-4} and train NoMaD for 30 epochs with a batch size of 256. We use cosine scheduling and warmup to stabilize the training process and follow other hyperparameters from ViNT [242]. For the ViNT observation encoder, we use EfficientNet-B0 [261] to tokenize observations and goals into

256-dimensional embeddings, followed by a Transformer decoder with 4 layers and 4 heads.

8.5 EVALUATION

We evaluate NoMaD in 6 distinct indoor and outdoor environments, and formulate our experiments to answer the following questions:

Q1. How does NoMaD compare to prior work for visual exploration and goal-reaching in real-world environments?

Q2. How does a joint task-agnostic and task-specific policy compare to the individual behavior policies?

Q3. How important is the choice of visual encoder and goal masking to the performance of NoMaD?

8.5.1 *Benchmarking Performance*

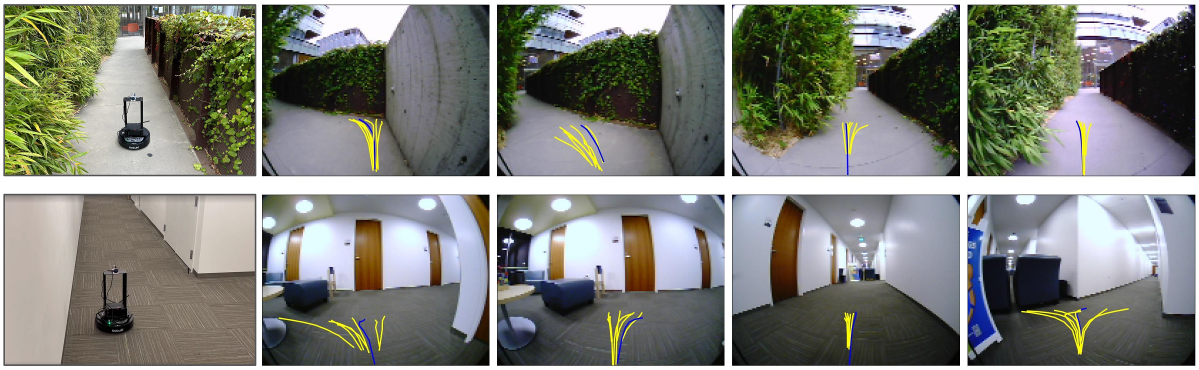


Figure 47: Visualizing rollouts of NoMaD deployed in challenging indoor (top) and outdoor (bottom) environments on the LoCoBot platform, showcasing successful exploration trajectories. Future action samples from the undirected mode are shown in yellow, and the action selected by the high-level planner is shown in blue. The predicted actions follow implicit navigational affordances, such as following hallways, and become multimodal at decision points, such as intersections in the hallway.

Towards understanding **Q1**, we compare NoMaD to six performant baselines for exploration and navigation in 6 challenging real-world environments. We follow the experimental setup of ViNT [242] and evaluate the methods on their ability to (i) explore a novel environment effectively in search of a goal position, or (ii) reach a goal indicated by an image in a previously explored environment, where the robot uses the policy to create a topological graph as episodic memory. All baselines are trained on a combination of GNM and SACSoN datasets for 20 epochs, and we perform minimal hyperparameter tuning to ensure stable training for each baseline. We report the mean success rate for each baseline, as well as the mean number of collisions per experiment.

VIB: We use the authors' implementation of a latent goal model for exploration [238], which uses a variational information bottleneck (VIB) to model a distribution of actions conditioned on observations.

MASKED VINT: We integrate our goal masking with the ViNT policy [242] to flexibly condition on the observation context c_t. This baseline predicts point estimates of future actions conditioned on c_t, rather than modeling the distribution.

AUTOREGRESSIVE: This baseline uses autoregressive predictions over a discretized action space to better represent multimodal action distributions. Our implementation uses a categorical representation of the action distribution, goal masking, and the same visual encoder design.

SUBGOAL DIFFUSION: We use the authors' implementation of the ViNT system [242] that pairs a goal-conditioned policy with an image diffusion model for generating candidate subgoal images, which are used by the policy to predict exploration actions. This is the best published baseline we compare against, but uses a 15× larger model than NoMaD.

RANDOM SUBGOALS: A variation of the above ViNT system which replaces subgoal diffusion with randomly sampling the training data for a candidate subgoal, which is passed to the goal-conditioned policy to predict exploration actions. This baseline does not use image diffusion, and has comparable parameter-count to NoMaD.

Table 4 summarizes the results of our experiments in 5 challenging indoor and outdoor environments. VIB and Masked ViNT struggle in all the environments we tested and frequently end in collisions, likely due to challenges with effectively modeling multimodal action distributions. The Autoregressive baseline uses a more expressive policy class and outperforms these baselines, but struggles in complex environments. Furthermore, the deployed policy tends to be jerky and slow to respond to dynamic obstacles in the environment, likely due to the discretized action space (see supplemental video for experiments). NoMaD consistently outperforms all baselines and results in smooth, reactive policies. For exploratory goal discovery, NoMaD outperforms the best published baseline (Subgoal Diffusion) by over 25% in terms of both efficiency and collision avoidance, and succeeds in all but the hardest environment. For navigation in known environments, using a topological graph, NoMaD matches the performance of the best published baseline, while also requiring a 15× smaller model and running entirely

		Exploration		Navigation
Method	Params	Success	Coll.	Success
Masked ViNT^m	15M	50%	1.0	30%
VIB [238]	6M	30%	4.0	15%
Autoregressive^m	19M	90%	2.0	60%
Random Subgoals [242]	30M	70%	2.7	**90%**
Subgoal Diffusion [242]	335M	77%	1.7	**90%**
NoMaD	19M	**98%**	**0.2**	**90%**

Table 4: NoMaD paired with a topological graph consistently outperforms all baselines for the task of exploration in previously unseen environments, and navigation in known environments. Most notably, NoMaD outperforms the state-of-the-art (Subgoal Diffusion) by 25%, while also avoiding collisions and requiring 15× fewer parameters. ^{m}These baselines that use goal masking.

Method	Params	Undirected	Goal-Conditioned
Diffusion Policy	15M	98%	✗
BESO [212]	15M	✗	94%
ViNT Policy	16M	✗	92%
NoMaD	19M	98%	92%

Table 5: Despite having comparable model capacities, NoMaD matches the performance of the best individual behavior policies for undirected exploration and goal-conditioned navigation.

on-the-edge. Figure 47 shows example rollouts of the NoMaD policy exploring unknown indoor and outdoor environments in search for the goal.

Analyzing the policy predictions across baselines (see Figure 48), we find that while the Autoregressive policy representation can (in principle) express multimodal distributions, the predictions are largely unimodal, equivalent to the policy learning the *average* action distribution. The Subgoal Diffusion baseline tends to represent the multiple modes well, but is not very robust. NoMaD consistently captures the multimodal distribution, and also makes accurate predictions when conditioned on a goal image.

8.5.2 *Unified v/s Dedicated Policies*

With the flexibility that a policy with task-specific and task-agnostic capabilities offers, **Q2** aims to understand the impact of goal masking on the individual behaviors learned by the policy. Specifically, we compare the performance of the jointly trained NoMaD model to the best-performing goal-conditioned and undirected models. We report the mean success rate for each baseline.

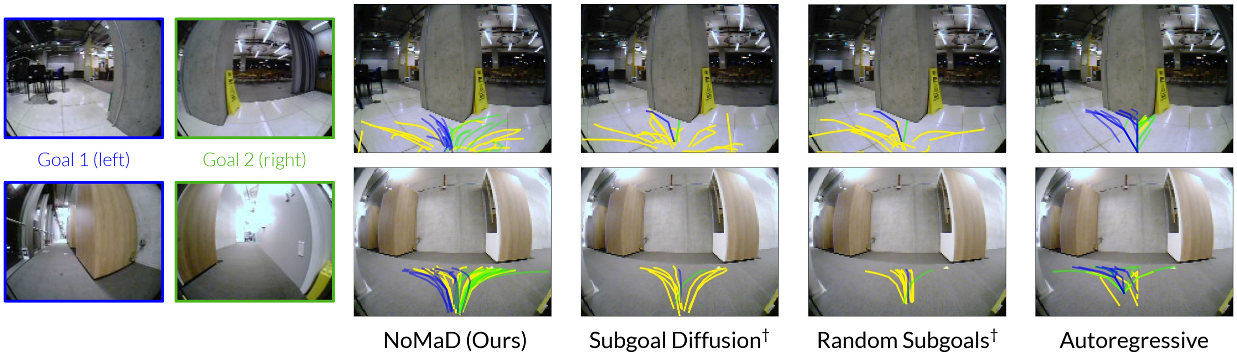


Figure 48: Examples of action predictions from NoMaD and baselines in undirected mode (yellow) and goal-directed mode with two different goal images (blue towards left, green towards right). Only NoMaD can consistently represent multimodal undirected predictions while avoiding collisions with pillars or walls, as well as correctly predicting the goal-conditioned action predictions for the two goals. †Note that Subgoal Diffusion and Random Subgoals baselines only represent point estimates when conditioned on a goal image.

DIFFUSION POLICY: We train a diffusion policy [42] with the same visual encoder as NoMaD and $m = 0$. This is the best exploration baseline, outperforming both VIB and IBC.

VINT POLICY: We use the authors' published checkpoint of the ViNT navigation policy [242], which predicts point estimates of future actions conditioned on observations and a goal. This is the best navigation baseline.

Comparing the unified NoMaD policy to the above, we find that despite having comparable model capacities, the unified policy trained with goal masking matches the performance of ViNT policy for goal-conditioned navigation and DP for undirected exploration. This suggests that training for these two behaviors involves learning shared representations and affordances, and a single policy can indeed excel at both task-agnostic and task-oriented behaviors simultaneously.

8.5.3 *Visual Encoder and Goal Masking*

We explore variations of the visual encoder and goal masking architectures to understand **Q3**. We consider two alternative visual encoder designs based on CNN and ViT backbones, and implement goal masking in different ways. We report the mean success rate for each baseline, as well as the mean number of collisions per experiment.

EARLY/LATE FUSION CNN: We use convolutional encoders followed by an MLP to encode the observation and goal images, and use dropout on the goal embeddings

Visual Encoder	Success	# Collisions
Late Fusion CNN	52%	3.2
Early Fusion CNN	68%	1.5
ViT	32%	2.5
NoMaD	**98%**	**0.2**

Table 6: The performance of NoMaD depends on the choice of visual encoder and goal masking strategy. The ViNT encoder with attention-based goal masking outperforms all alternatives.

followed by another MLP block to flexibly condition the observation context c_t on the goal. c_t obtained after dropout is used for conditioning the diffusion model in the same manner as NoMaD. We use a straight-through estimator [15] for propagating gradients to the observation and goal encoders during training. The goal can be combined with the observations either before or after the final MLP layers.

VIT: We divide the observation and goal images into 6×6 patches, and encode them using a Vision Transformer [60] into observation context c_t. We use attention masks to block the goal patches from propagating information downstream.

We find the choice of visual encoder to be crucial for training diffusion policies, as summarized in Table 6. NoMaD outperforms both the ViT- and CNN-based architectures, successfully reaching the goal while avoiding collisions. CNN with early fusion outperforms late fusion, confirming similar analysis in prior work [190, 242], but struggles to effectively condition on goal information. Despite it's high capacity, the ViT encoder struggled learn a good policy, likely due to optimization challenges in training end-to-end with diffusion.

8.6 DISCUSSION

We presented NoMaD, the first instantiation of a goal-conditioned diffusion policy, that can perform both task-agnostic exploration and task-oriented navigation. Our unified navigation policy uses a high-capacity Transformer encoder with masked attention approach to flexibly condition on the task, such as goal images for navigation, and models the actions conditioned on observations using a diffusion model. We study the performance of this unified model in the context of long-horizon exploration and navigation in previously unseen indoor and outdoor environments, demonstrating over 25% improvements in performance over the state-of-the-art in previously unseen settings, while also requiring 15× fewer computational resources.

While our experiments provide a proof of concept that unified policies can provide more effective navigation in new environments, our system has a number of limitations that could be addressed in future work. The navigation tasks are specified via goal images which, though quite general, are sometimes not the most natural modalities for users to employ. Extending our approach into a complete navigation system that can accommodate a variety of goal modalities, including language and spatial coordinates, would make our approach more broadly applicable. Additionally, our exploration method uses a standard frontier-based exploration strategy for high-level planning, leveraging our policy to explore at the frontier. Intelligently selecting which regions to explore, such as strategies based on semantics and prior knowledge, could further improve performance. We hope that these directions would lead to even more practical and capable systems, enabled by our policy representation.

acknowledgments

This research was partly supported by ARL DCIST CRA W911NF-17-2-0181, ARO W911NF-21-1-0097, IIS-2150826, and compute from Google TPU Research Cloud, NSF CloudBank, and the Berkeley Research Computing program. The authors would like to thank Kyle Stachowicz for useful discussions in early stages of the project, and Cheng Chi for help with reproducing the diffusion policy. The authors would also like to thank Michael Equi, Nathaniel Simon, and Anirudha Majumdar, for testing and providing feedback on preliminary versions of the NoMaD model.

Part III

COMBINING ROBOT AND INTERNET FOUNDATION MODELS

9

NAVIGATION WITH FOUNDATION MODELS OF LANGUAGE, VISION, AND ACTION

Synopsis

Goal-conditioned policies for robotic navigation can be trained on large, unannotated datasets, providing for good generalization to real-world settings (Parts I & II). However, particularly in vision-based settings where specifying goals requires an image, this makes for an unnatural interface. Language provides a more convenient modality for communication with robots, but contemporary methods typically require expensive supervision, in the form of trajectories annotated with language descriptions. We present a system, LM-Nav, for robotic navigation that enjoys the benefits of training on unannotated large datasets of trajectories, while still providing a high-level interface to the user. Instead of utilizing a labeled instruction following dataset, we show that such a system can be constructed entirely out of pre-trained foundation models for navigation (ViNG, GNM), image-language association (CLIP), and language modeling (GPT-3), without requiring any fine-tuning or language-annotated robot data. LM-Nav extracts landmarks names from an instruction, grounds them in the world via the image-language model, and then reaches them via the (vision-only) navigation model. We instantiate LM-Nav on a real-world mobile robot and demonstrate long-horizon navigation through complex, outdoor environments from natural language instructions.

9.1 INTRODUCTION

One of the central challenges in robotic learning is to enable robots to perform a wide variety of tasks on command, following high-level instructions from humans. This requires robots that can understand human instructions, and are equipped with a large

Shah†*, Osinski*†*, Ichter, Levine, "LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action", in Annual Conference on Robot Learning (CoRL) 2022*
Project website: sites.google.com/view/lmnav

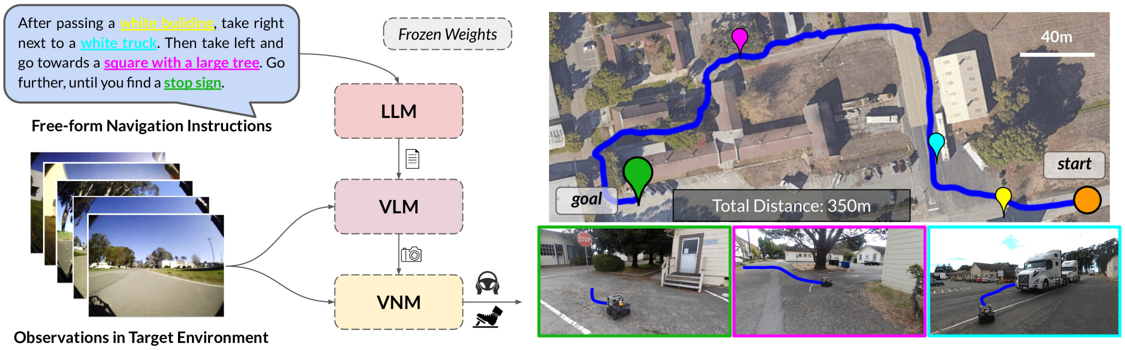

After passing a white building, take right next to a white truck. Then take left and go towards a square with a large tree. Go further, until you find a stop sign.
Free-form Navigation Instructions
Observations in Target Environment
Frozen Weights
LLM
VLM
VNM
40m
start
goal
Total Distance: 350m

Figure 49: Embodied instruction following with LM-Nav: Our system takes as input a set of raw observations from the target environment and free-form textual instructions (left), deriving an actionable plan using three *pre-trained* models: a large language model (**LLM**) for extracting landmarks, a vision-and-language model (**VLM**) for grounding, and a visual navigation model (**VNM**) for execution. This enables LM-Nav to follow textual instructions in complex environments purely from visual observations (right) *without any fine-tuning*.

repertoire of diverse behaviors to execute such instructions in the real world. Prior work on instruction following in navigation has largely focused on learning from trajectories annotated with textual instructions [9, 86, 142, 113, 292]. This enables understanding of textual instructions, but the cost of data annotation impedes wide adoption. On the other hand, recent work has shown that learning robust navigation is possible through goal-conditioned policies trained with self-supervision. These utilize large, unlabeled datasets to train vision-based controllers via hindsight relabeling [169, 251, 80, 139, 123, 241]. They provide scalability, generalization, and robustness, but usually involve a clunky mechanism for goal specification, using locations or images. In this work, we aim to combine the strengths of both approaches, enabling a robotic navigation system to execute natural language instructions by leveraging the capabilities of pre-trained models *without any user-annotated navigational data*. Our method uses these models to construct an "interface" that humans can use to communicate desired tasks to robots. This system enjoys the impressive generalization capabilities of the pre-trained language and vision-language models, enabling the robotic system to accept complex high-level instructions.

Our main observation is that we can utilize off-the-shelf *pre-trained models* trained on large corpora of visual and language datasets — that are widely available and show great few-shot generalization capabilities — to create this interface for embodied instruction following. To achieve this, we combine the strengths of two such robot-agnostic pre-trained models with a pre-trained navigation model. We use a visual navigation model (**VNM**: ViNG [241] or GNM [234]) to create a topological "mental map" of the environment using the robot's observations from a prior exploration of the environment. Given free-form textual instructions, we use a pre-trained large language model (**LLM**:

GPT-3 [22]) to decode the instructions into a sequence of textual landmarks. We then use a vision-language model (**VLM**: CLIP [205]) for *grounding* these textual landmarks in the topological map, by inferring a joint likelihood over the landmarks and nodes. A novel search algorithm is then used to plan a path for the robot, which is then executed by **VNM**. While reducing the task of language following to a combination of grounding and subgoal selection discards a lot of useful cues such as relations and verbs, we find that it is still sufficient to follow a variety of natural language instructions.

Our primary contribution is **L**arge **M**odel **Nav**igation, or LM-Nav, an embodied instruction following system that combines three large independently pre-trained models — a robotic control model that utilizes visual observations and physical actions (**VNM**), a vision-language model that grounds images in text but has no context of embodiment (**VLM**), and a large language model that can parse and translate text but has no sense of visual grounding or embodiment (**LLM**) — to enable long-horizon instruction following in complex, real-world environments. *We present the first instantiation of a robotic system that combines the confluence of pre-trained vision-and-language models with a goal-conditioned controller, to derive actionable plans* without any fine-tuning *in the target environment.* Notably, all three models are trained on large-scale datasets, with self-supervised objectives, and used off-the-shelf with *no fine-tuning* — no human annotations of the robot navigation data are necessary to train LM-Nav. We show that LM-Nav is able to successfully follow natural language instructions in pre-explored environments over the course of 100s of meters of complex, suburban navigation, while disambiguating paths with fine-grained commands.

9.2 RELATED WORK

Early works in augmenting navigation policies with natural language commands use statistical machine translation [134] to discover data-driven patterns to map free-form commands to a formal language defined by a grammar [285, 172, 35, 263, 173]. However, these approaches tend to operate on structured state spaces. Our work is closely inspired by methods that instead reduce this task to a sequence prediction problem [243, 176, 9]. Notably, our goal is similar to the task of VLN — leveraging fine-grained instructions to control a mobile robot solely from visual observations [9, 86].

However, most recent approaches to VLN use a large dataset of simulated trajectories — over 1M demonstrations — annotated with fine-grained language labels in indoor [142, 113, 292, 9, 245] and driving scenarios [36, 95, 182, 274, 183, 18], and rely on sim-to-real transfer for deployment in simple indoor environments [140, 8]. However, this necessitates building a photo-realistic simulator resembling the target environment, which can be challenging for unstructured environments, especially for the task of outdoor navigation. Instead, LM-Nav leverages free-form textual instructions to navigate a robot in complex, outdoor environments *without* access to any simulation or any trajectory-level annotations.

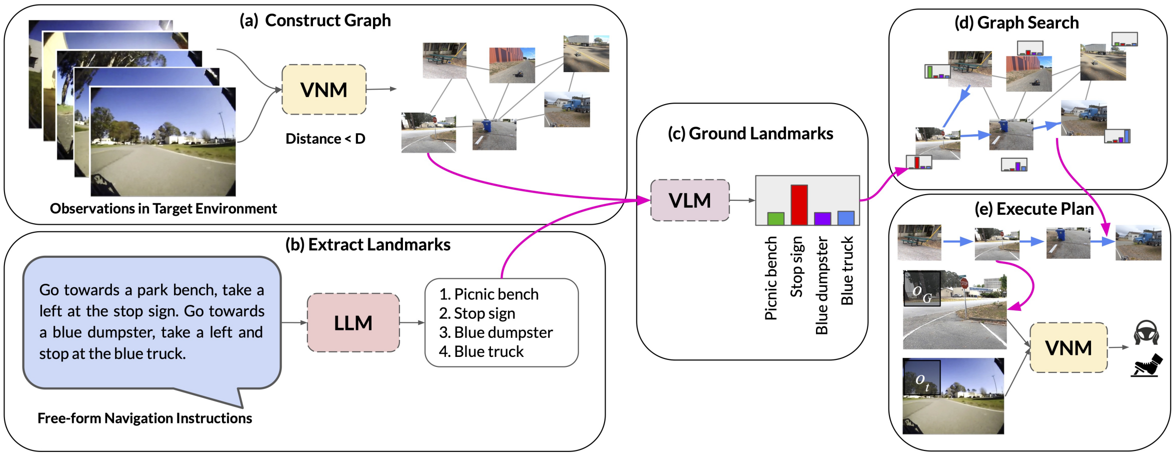


Figure 50: System overview: (a) **VNM** uses a goal-conditioned distance function to infer connectivity between the set of raw observations and constructs a topological graph. (b) **LLM** translates natural language instructions into a sequence of textual landmarks. (c) **VLM** infers a joint probability distribution over the landmark descriptions and nodes in the graph, which is used by (d) a graph search algorithm to derive the optimal walk through the graph. (e) The robot drives following the walk in the real world using the **VNM** policy.

Recent progress in using large-scale models of natural language and images trained on diverse data has enabled applications in a wide variety of textual [284, 264, 38], visual [205, 209, 221, 87, 118, 252], and embodied domains [244, 114, 109, 4, 299, 130]. In the latter category, approaches either fine-tune embeddings from pre-trained models on robot data with language labels [244, 130, 114], assume that the low-level agent can execute textual instructions (without addressing control) [109], or assume access to a set of text-conditioned skills that can follow atomic textual commands [4]. All of these approaches require access to low-level skills that can follow rudimentary textual commands, necessitating language annotations for robotic experience and a strong assumption on the robot's capabilities. In contrast, we combine these pre-trained vision and language models with pre-trained visual policies that do not use any language annotations [241, 238] *without* fine-tuning these models for the task of VLN.

LM-Nav uses self-supervised policies trained in a large number of prior environments, augmented with pre-trained vision and language models for parsing natural language instructions, and deploys them in novel real-world environments *without* any fine-tuning. We emphasize that while LM-Nav relies on a pre-built topological graph (Chapters 2, 6), this assumption may be relaxed by incorporating exploration heuristics in unseen environments (Chapters 4, 7), and can be an interesting avenue for future work.

9.3 LM-Nav: Instruction Following with Pre-trained Models

LM-Nav combines the components discussed earlier to follow natural language instructions in the real world. The **LLM** parses free-form instructions into a list of landmarks $\bar{\ell}$ (Sec. 9.3.2), the **VLM** associates these landmarks with nodes in the graph by estimating the probability that each node $\bar{v}$ corresponds to each $\bar{\ell}$, $P(\bar{v}|\bar{\ell})$ (Sec. 9.3.3), and the **VNM** is used to infer how effectively the robot can navigate between each pair of nodes in the graph, denoted by a probability $P(\overline{v_i, v_j})$. To find the optimal "walk" on the graph that both (i) adheres to the provided instructions and (ii) minimizes traversal cost, we derive a probabilistic objective (Sec. 9.3.1) and show how it can be optimized using a graph search algorithm (Sec. 9.3.4). This walk is executed in the real world by the **VNM** model.

9.3.1 *Problem Formulation*

Given a sequence of landmark descriptions $\bar{\ell} = \ell_1, \ell_2, ..., \ell_n$ extracted from the language command, our method needs to determine a sequence of waypoints $\bar{v} = v_1, v_2, ..., v_k$ to command to the robot. Typically, $k \geq n$, since each landmark needs to be visited, but the traversal might require other waypoints in between the landmarks. Finding $\bar{v}$ can formulated as a probabilistic inference problem. A key element in this formulation is access to a distribution $p(v_i|\ell_j)$ for each graph vertex v_i and landmark description ℓ_j. Recall that the graph vertices correspond to images observed by the robot, and thus, $p(v_i|\ell_i)$ represents a distribution over images given a language description. This can be obtained from the **VLM**. Intuitively, the full likelihood that we need to optimize to determine the robot's plan will now depend on two terms: likelihoods of the form $p(v_{t_i}|\ell_i)$ that describe how likely v_{t_i} is to correspond to ℓ_i for an assignment $t_1, t_2, \ldots, t_n$, and traversability likelihoods $p(\overline{v_i, v_{i+1}})$ that describe how likely is the robot to be able to reach v_{i+1} from v_i.

While we can use a variety of traversability likelihood functions, a simple choice is to use a discounted Markovian model, where the discount γ models the probability of *exiting* at each time step, leading to a termination probability of $1 - \gamma$ at each step, and a probability of reaching v_{i+1} given by $\gamma^{D(v_i, v_{i+1})}$, where $D(v_i, v_{i+1})$ is the estimated number of time steps the robot needs to travel from v_i to v_{i+1}, which is predicted by the **VNM**. While other traversability likelihoods could also be used, this choice is a convenient consequence of goal-conditioned reinforcement learning formulations [122, 91], and thus, the log-likelihood corresponds to $D(v_i, v_{i+1})$. We can use these likelihoods to derive the probability that a given sequence $\bar{v}$ can be traversed successfully, which we denote with the auxiliary Bernoulli random variable $c_{\bar{v}}$ (i.e., $c_{\bar{v}} = 1$ implies that $\bar{v}$ was traversed successfully):

$$P(c_{\bar{v}} = 1|\bar{v}) = \prod_{1 \leq i < T} P(\overline{v_i, v_{i+1}}) = \prod_{1 \leq i < T} \gamma^{D(v_i, v_{i+1})}, \tag{10}$$

The full likelihood used for planning is then given by:

$$P(\text{success}|\bar{v},\bar{\ell}) \propto P(c_{\bar{v}}=1|\bar{v})P(\bar{v}|\bar{\ell}) \; = \prod_{1\leq j<k} \gamma^{D(v_j,v_{j+1})} \max_{1\leq t_1\leq \ldots \leq t_n \leq k} \prod_{1\leq i\leq n} P(v_{t_i}|\ell_i). \quad (11)$$

9.3.2 *Parsing Free-Form Textual Instructions*

The user specifies the route they want the robot to take using natural language, while the objective above is defined in terms of a sequence of desired landmarks. To extract this sequence from the user's natural language instruction we employ a large language model, which in our prototype is GPT-3 [22]. We used a prompt with 2 examples of correct landmarks' extractions, followed by the description to be translated by the **LLM**. Examples of instructions and landmarks extracted by the model can be found in Fig. 52. The prompt was selected to disambiguate nuanced cases, e.g. when order of landmarks in the text is different than in the expected path (see example in Fig. 52 a). For details of the *"prompt engineering"* please see Appendix E.1.

9.3.3 *Visually Grounding Landmark Descriptions*

Algorithm 8 Graph Search

```
Input: Landmarks $(\ell_1, \ell_2, \ldots, \ell_n)$.
Input: Graph $\mathcal{G}(V, E)$.
Input: Starting node $S$.
$\forall_{\substack{i=0,\ldots,n \\ v\in V}} \; Q[i,v] = -\infty$
$Q[0,S] = 0$
Dijkstra_algorithm($\mathcal{G}, Q[0,*]$)
for $i$ in $1,2,\ldots,n$ do
    $\forall v \in V \; Q[i,v] = Q[i-1,v] + \text{CLIP}(v,\ell_i)$
    Dijkstra_algorithm($\mathcal{G}, Q[i,*]$)
end for
destination = $\arg\max(Q[n,*])$
return backtrack(destination, $Q[n,*]$)
```

As discussed in Sec. 9.3.1, a crucial element of selecting the walk through the graph is computing $P(v_i|\ell_j)$, the probability that landmark description v_i refers to node ℓ_j (see Eqn. 11). With each node containing an image taken during initial data collection, the probability can be computed using CLIP [205] in the way described in Sec. 9.4 as the retrieval task. As presented in Fig. 51, we apply CLIP to the image at node v_i and caption prompt in the form of *"This is a photo of a [ℓ_j]"*. To go from CLIP model outputs, which are logits, to probabilities we use $P(v_i|\ell_j) = \frac{\exp \text{CLIP}(v_i,\ell_j)}{\sum_{v\in V} \exp \text{CLIP}(v,\ell_j)}$. The resulting probability $P(v_i|\ell_j)$, together with the inferred edges' distances will be used to select the optimal walk.

9.3.4 *Graph Search for the Optimal Walk*

As described in Sec. 9.3.1, LM-Nav aims at finding a walk $\bar{v} = (v_1, v_2, \ldots, v_k)$ that maximizes the probability of successful execution of $\bar{v}$ that adheres to the given list of landmarks $\bar{\ell}$. We can define a function $R(\bar{v}, \bar{t})$ for a monotonically increasing sequence of indices $\bar{t} = (t_1, t_2, \ldots, t_n)$:

$$R(\bar{v}, \bar{t}) := \sum_{i=1}^{n} \text{CLIP}(v_{t_i}, \ell_i) - \alpha \sum_{j=1}^{T-1} D(v_j, v_{j+1}), \text{where}\, \alpha = -\log \gamma. \tag{12}$$

R has the property that $(\bar{v})$ maximizes $P(\text{success}|\bar{v}, \bar{\ell})$ defined in Eqn. 11, if and only if there exists $\bar{t}$ such that $(\bar{v}, \bar{t})$ maximizes R. In order to find such $(\bar{v}, \bar{t})$, we employ dynamic programming. In particular we define a helper function $Q(i, v)$ for $i \in \{0, 1, \ldots, n\}$, $v \in V$:

$$Q(i, v) = \max_{\substack{\bar{v}=(v_1,v_2,\ldots,v_j), v_j=v \\ \bar{t}=(t_1,t_2,\ldots,t_i)}} R(\bar{v}, \bar{t}). \tag{13}$$

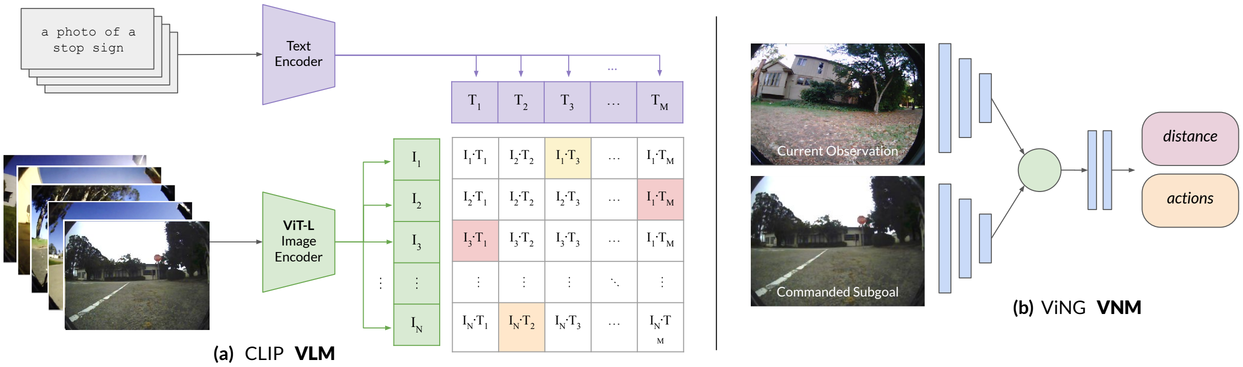
$Q(i, v)$ represents the maximal value of R for a walk ending in v that visited the landmarks up to index i. The base case $Q(0, v)$ visits none of the landmarks, and its value of R is simply equal to minus the length of shortest path from the starting node S. For $i > 0$ we have:

$$Q(i, v) = \max \left(Q(i-1, v) + \text{CLIP}(v, \ell_i), \max_{w \in \text{neighbors}(v)} Q(i, w) - \alpha \cdot D(v, w) \right). \tag{14}$$

The base case for DP is to compute $Q(0, V)$. Then, in each step of DP $i = 1, 2, \ldots, n$ we compute $Q(i, v)$. This computation resembles the Dijkstra algorithm ([55]). In each iteration, we pick the node v with the largest value of $Q(i, v)$ and update its neighbors based on the Eqn. 14. Algorithm 8 summarizes this search process. The result of this algorithm is a walk $\bar{v} = (v_1, v_2, \ldots, v_k)$ that maximizes the probability of successfully carrying out the instruction. Such a walk can be executed by **VNM**, using its action estimates to sequentially navigate to these nodes.

9.4 PRELIMINARIES

LM-Nav consists of three large, pre-trained models for processing language, associating images with language, and visual navigation.



(a) CLIP **VLM**

(b) ViNG **VNM**

Figure 51: LM-Nav uses CLIP to infer a joint distribution over textual landmarks and image observations. **VNM** infers a goal-conditioned distance function and policy that can control the robot.

Large language models are generative models of text trained on large corpora of internet text using self-supervised learning. LM-Nav uses the GPT-3 **LLM** [22] to parse instructions into a sequence of landmarks.

Vision-and-language models refer to models that can associate images and text, e.g. image captioning, visual question-answering, etc. [5, 157, 41]. We use the CLIP **VLM** [205], a model that jointly encodes images and text into a shared embedding space, to jointly encode a set of landmark descriptions t obtained from the **LLM** and a set of images i_k to obtain their **VLM** embeddings $\{T, I_k\}$ (see Fig. 50). Computing the cosine similarity between these embeddings, followed by a softmax operation results in probabilities $P(i_k|t)$, corresponding to the likelihood that image i_k corresponds to the string t. LM-Nav uses this probability to align landmark descriptions with images.

Visual navigation models learn navigational affordances directly from visual observations [224, 28, 96, 283, 241], associating images and actions through time. We use the ViNG **VNM** [241], a goal-conditioned model that predicts temporal distances between pairs of images and the corresponding actions to execute (see Fig. 50). The **VNM** serves two purposes: (i) given a set of observations in the target environment, the distance predictions from the **VNM** can be used to construct a topological graph $\mathcal{G}(V, E)$ that represents a "mental map" of the environment; (ii) given a "walk" (i.e., a sequence of connected subgoals to the goal), **VNM** can control the robot along this plan. The topological graph $\mathcal{G}$ is an important abstraction that allows a simple interface for planning over past experience in the environment and has been successfully used in prior work to perform long-horizon navigation [232, 179, 23]. To deduce connectivity in $\mathcal{G}$, we use a combination of learned distance estimates, temporal proximity (during data collection), and spatial proximity (using GPS measurements). For more details on the construction of this graph, see Appendix E.2.

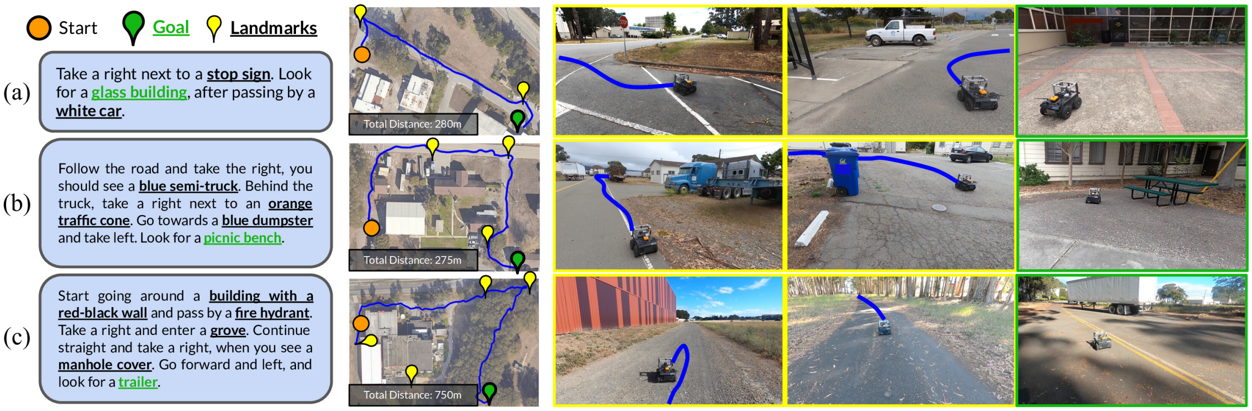

Start
Goal
Landmarks
(a)
Take a right next to a **stop sign**. Look for a **glass building**, after passing by a **white car**.
Total Distance: 280m
(b)
Follow the road and take the right, you should see a **blue semi-truck**. Behind the truck, take a right next to an **orange traffic cone**. Go towards a **blue dumpster** and take left. Look for a **picnic bench**.
Total Distance: 275m
(c)
Start going around a **building with a red-black wall** and pass by a **fire hydrant**. Take a right and enter a **grove**. Continue straight and take a right, when you see a **manhole cover**. Go forward and left, and look for a **trailer**.
Total Distance: 750m

Figure 52: Qualitative examples of LM-Nav in real-world environments executing textual instructions (left). The landmarks extracted by **LLM** (highlighted in text) are grounded into visual observations by **VLM** (center; overhead image not available to the robot). The resulting *walk* of the graph is executed by **VNM** (right).

9.5 SYSTEM EVALUATION

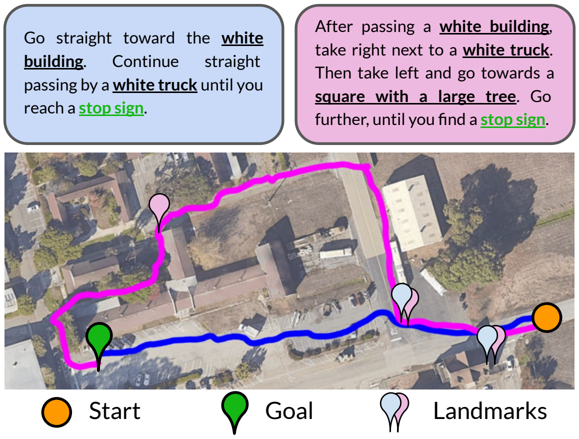
We now describe our experiments deploying LM-Nav in a variety of outdoor settings to follow high-level natural language instructions with a small ground robot (Clearpath Jackal UGV platform — see Fig. 49(right) for image and Appendix E.3 for details). For all experiments, the weights of **LLM**, **VLM**, and **VNM** are frozen — there is *no fine-tuning or annotation* in the target environment. We evaluate the complete system, as well as the individual components of LM-Nav, to understand its strengths and limitations. Our experiments demonstrate the ability of LM-Nav to follow high-level instructions, disambiguate paths, and reach goals that are up to 800m away.

9.5.1 *Following Instructions with LM-Nav*

In each evaluation environment, we first construct the graph by manually driving the robot and collecting image and GPS observations. The graph is constructed automatically using the **VNM** to predict relative distances between images in these trajectories. We tested our system on 20 queries in 2 environments, corresponding to a combined length of over 6km. The instructions include prominent landmarks that can be identified from the robot's observations, e.g., buildings and stop signs.

Fig. 52 shows qualitative examples of the path taken by the robot. In Fig. 52(a), LM-Nav is able to successfully localize the simple landmarks from its prior traversal and find a short path to the goal. While there are multiple stop signs in the environment, the objective in Eqn. 11 causes the robot to pick the correct one, minimizing overall trajectory length. Fig. 52(b) highlights LM-Nav's ability to follow complex instructions with multiple landmarks — despite the possibility of taking a shorter route directly to the final landmark, the robot follows a path that correctly visits all of the landmarks.

MISSING LANDMARKS. While LM-Nav is effective at finding a path through landmarks extracted from instructions, it relies on the assumption that the landmarks (i) exist in the environment, and (ii) can be identified by the **VLM**. Fig. 52(c) illustrates a case where the executed path fails to visit one of the landmarks — a fire hydrant — and takes a path that goes around the top of the building rather than the bottom. This failure mode is attributed to the the inability of the **VLM** to detect a fire hydrant from the robot's observations. On independently evaluating the efficacy of the **VLM** at retrieving landmarks (see Sec. 9.5.3), we find that despite being the best off-the-shelf model for our task, CLIP is unable to retrieve a small number of "hard" landmarks, including fire hydrants and cement mixers. In many practical cases, the robot is still successful in finding a path that visits the remaining landmarks.



Go straight toward the **white building**. Continue straight passing by a **white truck** until you reach a **stop sign**.

After passing a **white building**, take right next to a **white truck**. Then take left and go towards a **square with a large tree**. Go further, until you find a **stop sign**.

Figure 53: LM-Nav can successfully disambiguate instructions with same start-goal locations that differ slightly. The landmarks are underscored in text and their locations are marked with pins.

DISAMBIGUATION WITH INSTRUCTIONS. Since the objective of LM-Nav is to follow instructions, and not merely to reach the final goal, different instructions may lead to different traversals. Fig. 53 shows an example where modifying the instruction can disambiguate multiple paths to the goal. Given the shorter prompt (blue), LM-Nav prefers the more direct path. On specifying a more fine-grained route (magenta), LM-Nav takes an alternate path that passes a different set of landmarks.

9.5.2 *Quantitative Analysis*

To quantify the performance of LM-Nav, we introduce the following metrics. A walk found by the graph search is *successful*, if (1) it matches the path intended by the user or (2) if the landmark images extracted by the search algorithm contain said landmarks (i.e. if the path visits landmarks with the same description, even if not exactly the same). *Planning success* is the fraction of successful walks found by the search algorithm. *Efficiency* of a walk is defined as the ratio of the lengths of the described route and the executed one; the value is clipped at a maximum of 1 to account for the cases when the LM-Nav executes a path shorter than the user intended. For a set of queries, we report the average efficiency over successful experiments. The *planning efficiency* is similarly defined as the

System	Environment	Net Success ↑	Efficiency ↑	# Diseng. ↓	Planning ↑
GPS-Nav (No **VNM**)	EnvSmall-10	0.23	0.93	0.75	**0.9**
LM-Nav (Ours)	EnvSmall-10	**0.8**	**0.96**	**0.1**	**0.9**
	EnvLarge-10	**0.8**	**0.89**	**0**	**0.8**

Table 7: Quantifying navigational instruction following with LM-Nav over 20 experiments. LM-Nav can successfully plan a path to the goal, and follow it efficiently, over 100s of meters. Ablating the **VNM** (GPS-Nav) severely hurts performance due to frequent disengagements inability to reason about collisions with obstacles.

LLM Candidate	Avg. Extraction Success
Noun Chunks	0.88
fairseq-1.3B [11]	0.52
fairseq-13B [11]	0.76
GPT-J-6B [276]	0.80
GPT-NeoX-20B [17]	0.72
GPT-3 [22]	**1.0**

Table 8: GPT-3 consistently outperforms alternatives in parsing free-form instructions into landmarks.

VLM Candidate	Detection Rate
Faster-RCNN [211]	0.07
ViLD [87]	0.38
CLIP-ViT [205]	**0.87**

Table 9: CLIP-ViT produces the most reliable landmark detections from visual observations.

ratio of the length of the described and *planned* routes. Finally, *number of disengagements* is the average number of human interventions per experiment due to unsafe maneuvers.

Table 7 summarizes the quantitative performance of the system over 20 instructions. LM-Nav generates a successful walk for 85% of them, and causes disengagement only once (an average of 1 intervention per 6.4km of traversals). Investigating the planning failure modes suggests that the most critical component of our system is the ability of **VLM** to detect certain landmarks, e.g. a fire hydrant, and in challenging lighting conditions, e.g. underexposed images.

9.5.3 *Dissecting LM-Nav*

To understand the influence of each of the components of LM-Nav, we conduct experiments to evaluate these components in isolation. For more details about these experiments, see Appendix E.4.

To evaluate the performance of **LLM** candidates in parsing instructions into an *ordered* list of landmarks, we compare GPT-3 (used by LM-Nav) to other state-of-the-art pre-

	EnvSmall-10		EnvLarge-10	
Planner	Pl. Success ↑	Pl. Efficiency ↑	Pl. Success ↑	Pl. Efficiency ↑
Max Likelihood	0.6	0.69	0.2	0.17
LM-Nav (Ours)	**0.9**	**0.80**	**0.8**	**0.99**

Table 10: Ablating the search algorithm (Sec. 9.3.4) gives a max likelihood planner that ignores reachability information, resulting in inefficient plans that are up to 6× longer than LM-Nav for the same instruction.

trained language models — fairseq [11], GPT-J-6B [276], and GPT-NeoX-20B [17] — as well as a simple baseline using spaCy NLP library [102] that extracts base noun phrases, followed by filtering. In Table 8 we report the average extraction success for all the methods on the 20 prompts used in Section 9.5.2. GPT-3 significantly outperforms other models, owing to its superior representation capabilities and in-context learning [215]. The noun chunking performs surprisingly reliably, correctly solving many simple prompts. For further details on these experiments, see Appendix E.4.2.

To evaluate the **VLM**'s ability to ground these textual landmarks in visual observations, we set up an object detection experiment. Given an unlabeled image from the robot's on-board camera and a set of textual landmarks, the task is to *retrieve* the corresponding label. We run this experiment on a set of 100 images from the environments discussed earlier, and a set of 30 commonly-occurring landmarks. These landmarks are a combination of the landmarks retrieved by the **LLM** in our experiments from Sec. 9.5.1 and manually curated ones. We report the detection successful if any of the top 3 predictions adhere to the contents of the image. We compare the retrieval success of our **VLM** (CLIP) with some object detection alternatives — Faster-RCNN-FPN [211, 159], a state-of-the-art object detection model pre-trained on MS-COCO [160, 286], and ViLD [87], an open-vocabulary object detector based on CLIP and Mask-RCNN [93]. To evaluate against the closed-vocabulary baseline, we modify the setup by projecting the landmarks onto the set of MS-COCO class labels. We find that CLIP outperforms baselines by a wide margin, suggesting that its visual model transfers very well to robot observations (see Table 9). Despite deriving from CLIP, ViLD struggles with detecting complex landmarks like "manhole cover" and "glass building". Faster-RCNN is unable to detect common MS-COCO objects like "traffic light", "person" and "stop sign", likely due to the on-board images being out-of-distribution for the model.

To understand the importance of the **VNM**, we run an ablation experiment of LM-Nav without the navigation model. Using GPS-based distance estimates and a naïve straight line controller between nodes of the topological graph. Table 7 summarizes these results — without **VNM**'s ability to reason about obstacles and traversability, the system frequently runs into small obstacles such as trees and curbs, resulting in failure. Fig. 54

illustrates such a case — while such a controller works well on open roads, it fails to reason about connectivity around buildings or obstacles and results in collisions with a curb, a tree, and a wall in 3 individual attempts. This illustrates that using a learned policy and distance function from the **VNM** is critical for LM-Nav to successfully navigate in complex environments.

Lastly, to understand the importance of the two components of the graph search objective (Eqn. 12), we ran a set of ablations where the graph search only depends on $P(\bar{v}|\bar{\ell})$, i.e. *Max Likelihood Planning*, which only picks the most likely landmark without reasoning about topological connectivity or traversability. Table 10 shows that such a planner suffers greatly in the form of efficiency, because it does not utilize the spatial organization of nodes and their connectivity. For more details on these experiments, and qualitative examples, see Appendix E.4.

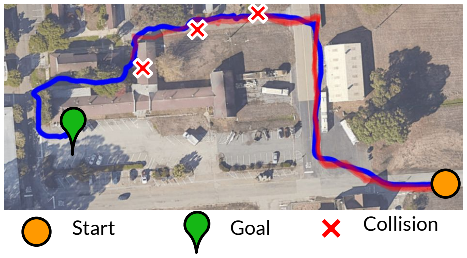


Figure 54: GPS-Nav (red) fails due to its inability to reason about traversability through obstacles, while LM-Nav (blue) succeeds.

9.6 Discussion

We presented **L**arge **M**odel **Nav**igation, a robotic system that can execute textual instructions in the real-world without requiring any human annotations for navigation trajectories. LM-Nav combines three pre-trained models: the **LLM**, which parses instructions into a list of landmarks; the **VLM**, which infers joint probabilities between these landmarks and visual observations from the environment; and the **VNM**, which estimates navigational affordances (distances between landmarks) and control actions. Each model is pre-trained on its own dataset, and we show that the complete system can execute a variety of user-specified instructions in real-world environments — choosing the correct sequence of landmarks by leveraging language and spatial context — and handle mistakes (such as missing landmarks). We also analyze the impact of each pre-trained model on the full system.

Limitations and future work The most prominent limitation of LM-Nav is its reliance on landmarks: while the user can specify any instruction they want, LM-Nav only focuses on the landmarks and disregards any verbs, propositions, adverbs, etc. (e.g., "go straight for three blocks" or "drive past the dog very slowly"), which can be lossy. Grounding such nuances is an important direction for future work. Additionally, LM-Nav uses a **VNM** that is specific to outdoor navigation with the Jackal robot, which limits wider adoption for other robot embodiments and sensor suites. An exciting direction for future work would be to swap in a "general navigation model" [234] that can be utilized broadly across robots, analogous to how the **LLM** and **VLM** handle any text or image. In its current form, LM-Nav provides a simple and attractive prototype for how pre-trained models can be combined to solve complex robotic tasks, and illustrates that these models can serve as an "interface" to robotic controllers that are trained without any language annotations. One of the implications of this result is that further progress on self-supervised robotic policies (e.g., goal-conditioned policies) can directly benefit instruction following systems. More broadly, understanding how modern pre-trained models enable effective decomposition of robotic control may enable broadly generalizable systems in the future, and we hope that LM-Nav will serve as a step in this direction.

ACKNOWLEDGMENTS

This research was supported by DARPA Assured Autonomy, DARPA RACER, Toyota Research Institute, ARL DCIST CRA W911NF-17-2-0181, and AFOSR. BO was supported by the Fulbright Junior Research Award granted by the Polish-U.S. Fulbright Commission. We would like to thank Alexander Toshev for pivotal discussions in early stages of the project. We would also like to thank Kuan Fang, Siddharth Karamcheti, and Albertyna Osińska for useful discussions and feedback.

10

SEMANTIC GUESSWORK AS A HEURISTIC FOR PLANNING

Synopsis

This chapter discusses another mechanism of using the inherent knowledge stored in Large Language Models (LLMs) for solving long-range planning tasks. Navigation in unfamiliar environments presents a major challenge for robots: while mapping and planning techniques can be used to build up a representation of the world, quickly discovering a path to a desired goal in unfamiliar settings with such methods often requires lengthy mapping and exploration. Humans can rapidly navigate new environments, particularly indoor environments that are laid out logically, by leveraging semantics — e.g., a kitchen often adjoins a living room, an exit sign indicates the way out, and so forth. Language models can provide robots with such knowledge, but directly using language models to instruct a robot how to reach some destination can also be impractical: while language models might produce a narrative about how to reach some goal, because they are not grounded in real-world observations, this narrative might be arbitrarily wrong. Therefore, in this paper we study how the "semantic guesswork" produced by language models can be utilized as a guiding heuristic for planning algorithms. Our method, Language Frontier Guide (LFG), uses the language model to bias exploration of novel real-world environments by incorporating the semantic knowledge stored in language models as a search heuristic for planning with either topological or metric maps. We evaluate LFG in challenging real-world environments and simulated benchmarks, outperforming uninformed exploration and other ways of using language models.

10.1 INTRODUCTION

Navigation in complex open-world environments is conventionally viewed as the largely geometric problem of determining collision-free paths that traverse the environment from

Shah†, Equi†, Osinski, Xia, Ichter, Levine, "Navigation with Large Language Models: Semantic Guesswork as a Heuristic for Planning", in Annual Conference on Robot Learning (CoRL) 2023
Project website: sites.google.com/view/lfg-nav

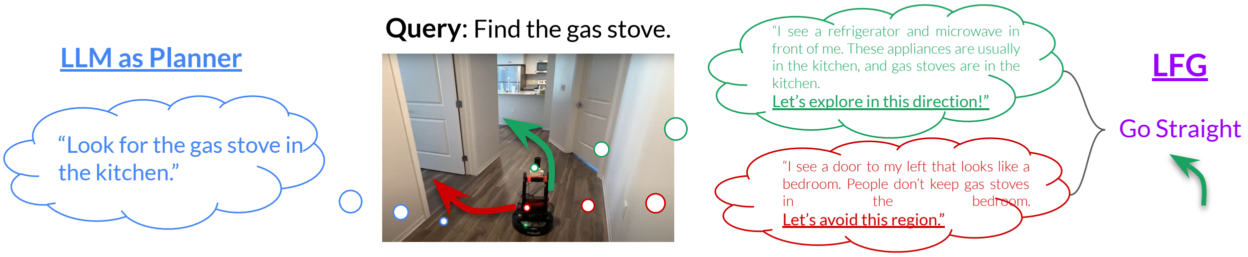

LLM as Planner
"Look for the gas stove in the kitchen."
Query: Find the gas stove.
"I see a refrigerator and microwave in front of me. These appliances are usually in the kitchen, and gas stoves are in the kitchen. Let's explore in this direction!"
"I see a door to my left that looks like a bedroom. People don't keep gas stoves in the bedroom. Let's avoid this region."
LFG
Go Straight

Figure 55: In constrast to methods that use LLM plans directly, Language Frontier Guide (LFG) uses a language model to *score* subgoal candidates, and uses these scores to guide a heuristic-based planner.

one location to another. However, real-world environments possess *semantics*. Imagine navigating an airport to get to a terminal: your prior knowledge about the way such buildings are constructed provides extensive guidance, even if this particular airport is unfamiliar to you. Large language models (LLMs) and various language embedding techniques have been studied extensively as ways to interpret the semantics in user-specified *instructions* (e.g., parsing "go to the television in the living room" and grounding it in a specific spatial location), but such models can provide much more assistance in robotic navigation scenarios by capturing rich semantic knowledge about the world. For instance, when looking for a spoon in an unseen house, the LLM can produce a "narrative" explaining why going towards a dishwasher may eventually lead you to find the spoon, and that the robot should prioritize that direction. This is similar to how a person might imagine different ways that the available subgoals might lie on the path to the goal, and start exploring the one for which this "narrative" seems most realistic. However, since language models are not *grounded* in the real world, such models do not know the spatial layout of the robot's surroundings (e.g., there is a couch that the robot needs to circumnavigate). To utilize the semantic knowledge in language models to aid in embodied tasks, we should not just blindly *follow* the language model suggestions, but instead use them as proposals or navigational heuristics. In this paper, we study how that might be accomplished.

We study this idea in the context of visual navigation, where a robot is tasked with reaching a goal denoted by a natural language query q (see Fig. 55) in a *novel* environment using visual observations. The robot has no prior experience in the target environment, and must explore the environment to look for the goal. While narratives generated by an LLM may not be sufficient for navigation by themselves, they contain useful cues that can be used to *inform* or *guide* the behavior of the underlying navigation stack for the language navigation task (e.g., by choosing between collision-free subgoal proposals that avoid the couch and lead to the ice tray). We show that this idea can be combined with frontier-based exploration, where the robot maintains a set of unvisited locations at its frontier, *grounds* them in text using a vision-language model (VLM), and *scores* the unvisited subgoals by using LLM reasoning.

We propose Language Frontier Guide, or LFG, a method for leveraging the reasoning capabilities of LLMs to produce a *search heuristic* for guiding exploration of previously unseen real-world environments, combining the strengths of search-based planning with LLM reasoning. LFG is agnostic of the memory representation and planning framework, and can be combined with both (i) a geometric navigation pipeline, building a metric map of the environment for planning and using a hand-designed controller, as well as (ii) a learning-based navigation pipeline, building a topological map for planning and using a learned control policy, yielding a versatile system for navigating to open-vocabulary natural language goals. Our experiments show that LFG can identify and predict simple patterns in previously unseen environments to accelerate goal-directed exploration. We show that LFG outperforms other LLM-based approaches for semantic goal-finding in challenging real-world environments and on the Habitat ObjectNav benchmark.

10.2 RELATED WORK

Learning-based approaches to navigation can exploit patterns in the training environments, particularly by learning vision-based navigation strategies through reinforcement or imitation [9, 225, 96, 234, 242, 208]. Our work is related to PONI [208], which uses a learned potential function to prioritize frontier points to explore; instead, we use a language model to rank these points. Notably, these methods do not benefit from prior semantic knowledge (e.g., from the web), and must rely entirely on patterns discovered from offline or online navigational data. Our aim is specifically to bring semantic knowledge into navigation, to enable robots to more effectively search for a goal in a new environment.

Prior knowledge about the semantics of indoor environments can provide significantly richer guidance. With the advent of effective open-vocabulary vision models [40, 205], some works have recently explored incorporating their semantic knowledge into models for navigation and other robotic tasks with the express aim of improving performance at *instruction following* [166, 168, 33, 105, 235]. In general within robotics, such methods have either utilized pre-trained vision-language representations [244, 231, 117], or used language models directly to make decisions [59, 175, 249, 287, 56, 158]. Our aim is somewhat different: while we also focus on language-specified goals, we are primarily concerned with utilizing the semantics in pre-trained language models to help a robot figure out how to actually reach the goal, rather than utilizing the language models to more effectively interpret a language instruction. While language models can output reasonable substeps for temporally extended tasks in some settings [110, 111], there is contradictory evidence about their ability to actually plan [272], and because they are unaware of the observations and layout in a particular environment, their "plans" depend entirely on the context that is provided to them. In contrast to prior work, our approach does not rely on the language model producing a *good* plan, but merely a heuristic that can bias a dedicated planner to reach a goal more effectively. In this way, we use the language models more to produce *suggestions* rather than actual plans.

Chapter 9 and similar works have sought to combine predictions from language models with either planning or probabilistic inference [108, 235], so as to not rely entirely on forward prediction from the language model to take actions. However, these methods are more aimed at filtering out *infeasible* decisions, for example by disallowing actions that a robot is incapable of performing, and still focus largely on being able to interpret and process instructions, rather than using the language model as a source of semantic hints. In contrast, by incorporating language model suggestions as heuristics into a heuristic planner, our approach can completely override the language model predictions if they are incorrect, while still making use of them if they point the way to the goal.

Another branch of recent research [119, 106, 61] has taken a different approach to ground language models, by making it possible for them to read in image observations directly. While this represents a promising alternative approach to make language models more useful for embodied decision making, we believe it is largely orthogonal and complementary to our work: although vision-language models can produce more grounded inferences about the actions a robot should take, they are still limited only to *guessing* when placed in unfamiliar environments. Therefore, although we use ungrounded language-only models in our evaluation, we expect that our method could be combined with vision-language models easily, and would provide complementary benefits.

10.3 PROBLEM FORMULATION AND OVERVIEW

Our objective is to design a high-level planner that takes as input a natural language query q (e.g., "find the bedside table"), explores the environment in search of the queried object, and commands a low-level policy to control a robotic agent. To do this, we maintain an episodic memory of the environment $\mathcal{M}$ in the form of either (i) a 2D map of the environment, where grid cells contain information about occupancy and semantic labels, or (ii) a topological map of the environment, where nodes contain images captured by the robot and corresponding object labels. One way to solve this task is Frontier-Based Exploration (FBE) [291], where a robot maintains a set of unexplored *frontiers* in it's memory, and explores randomly to reach the goal. *Can we do better with access to LLMs?*

We distill the language-guided exploration task to a heuristic-based search problem, where the robot must propose unvisited subgoals or waypoints, score them, and then use a search algorithm (e.g., A*) to plan a path to the goal. Thus, at the core of LFG is the task of *scoring* subgoal proposals. Formally, let's assume we have the task by query q, a partially explored environment stored in $\mathcal{M}$, and a set $\mathcal{S}$ of n textual subgoal proposals $s_1, s_2, \ldots, s_n$ (e.g., "a corner with a dishwasher and refrigerator", "a hallway with a door", etc.). Our goal is to score these subgoal proposals with $p(s_i, q, \mathcal{M})$, the probability that the candidate $s_i \in \mathcal{S}$ will lead to the goal q given the current state of the environment, described through $\mathcal{M}$.

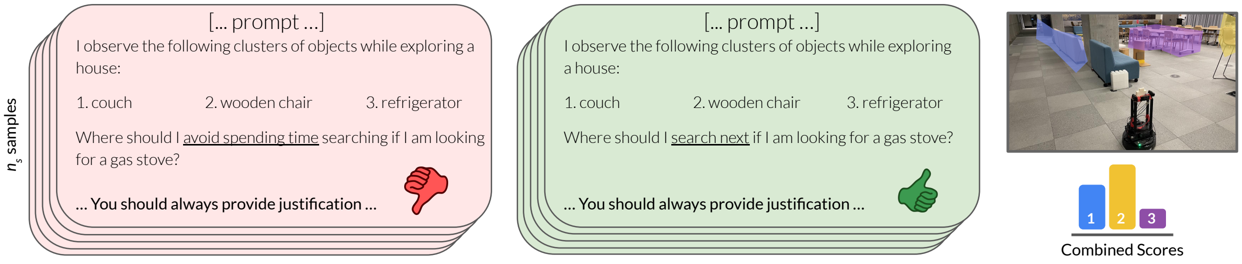

n_s samples
[... prompt ...]
I observe the following clusters of objects while exploring a house:
1. couch
2. wooden chair
3. refrigerator
Where should I avoid spending time searching if I am looking for a gas stove?
... You should always provide justification ...
[... prompt ...]
I observe the following clusters of objects while exploring a house:
1. couch
2. wooden chair
3. refrigerator
Where should I search next if I am looking for a gas stove?
... You should always provide justification ...
1
2
3
Combined Scores

Figure 56: LFG scores subgoals with an empirical estimate of the likelihoods by sampling an LLM n_s times with both positive and negative prompts, and uses chain-of-thought to obtain reliable scores. These scores are used by a high-level planner as *heuristics* to guide search. For full prompts, see Appendix F.2.

We posit that we can leverage the semantic reasoning capabilities of LLMs by prompting them to construct narratives about which semantic regions of the environment are most (and least) likely to lead to the goal. While the narrative itself might be ungrounded, since the LLM knows very little about the environment, reasoning over objects and semantic regions of the environment often generalizes very broadly. For example, even without seeing a new apartment, a human would *guess* that the dining area is close to the kitchen. Hence, rather than directly using LLM scores for planning [111, 158], we incorporate them as a goal-directed *heuristic* to inform the search process. This has two distinct advantages: (i) when the LLM is right, it nudges the search towards the goal, and when it is wrong (or uncertain), we can still default to the underlying FBE algorithm, allowing recovery from LLM failures, and (ii) it allows us to combine the signal from LLMs with other scores that may be more grounded, e.g. distance to subgoals, making the system more versatile.

10.4 LFG: SCORING SUBGOALS BY POLLING LLMS

Our aim in this section is to derive a scoring function from LLMs that takes a textual description of subgoal candidates s_i and the goal query q as inputs, and predicts task-relevant probability $p(s_i, q, \mathcal{M})$, conditioned on the episodic memory $\mathcal{M}$. While we may obtain this from next-token likelihoods (or "logprobs"), they do not represent the desired task-relevant probability $p(s_i, q, \mathcal{M})$, and fail to assign similar scores, say, to different subgoals that are semantically similar but have different tokenizations (see our experiments in Section 10.6 for a comparison). Furthermore, most capable LLMs of today are available through APIs that do not expose the ability to query logprobs.[1] And lastly, even if reliable logprobs were available, they are incompatible with chain-of-thought prompting [281], which we find to be crucial to success in spatial reasoning.

[1] Most notably, OpenAI's Chat API for GPT-3.5 and GPT-4, Google's PaLM API, and Anthropic's Claude API all do not support logprobs.

To overcome these challenges, LFG uses a novel approach to extract task-relevant likelihoods from LLMs. Given candidate subgoal images, LFG uses a VLM to obtain a textual subgoal desriptor s_i, which must be scored with the LLM. LFG *polls* the LLMs by sampling the most likely subgoal n_s times, conditioned on a task-relevant prompt. We then use these samples to empirically estimate the likelihood of each subgoal. To get informative and robust likelihood estimates, we use a chain-of-thought prompting (CoT) technique [281], to improve the quality and interpretability of the scores, and use a combination of positive and negative prompts to gather unbiased likelihood estimates. Figure 56 outlines our scoring technique, with the full prompt provided in Appendix F.2. We now describe the details of our scoring technique.

STRUCTURED QUERY: We rely on in-context learning by providing an example of a structured query-response pair to the LLM, and ask it to pick the most likely subgoal that satisfies the query. To sample a subgoal from $\mathcal{S}$ using a language model, we prompt it to generate a structured response, ending with "Answer: i". This structure allows us to always sample a *valid* subgoal, without having to ground LLM generations in the environment [110].

POSITIVES AND NEGATIVES: We find that only using positive prompts (e.g., "which subgoal is most likely to reach the goal") leads to likelihood estimates being uninformative for cases where the LLM is not confident about any subgoal. To overcome this, we also use negative prompts (e.g., "which subgoal is least likely to be relevant for the goal"), which allows us to score subgoals by eliminating/downweighting subgoals that are clearly irrelevant. We then use the difference between the positive and negative likelihoods to rank subgoals.

Algorithm 9 Scoring Subgoals with LFG

```
Input: Subgoal descriptors $\{l_i \forall s_i \in \mathcal{S}\}$
pPrompt ← PositivePrompt($\{l_i\}$)
nPrompt ← NegativePrompt($\{l_i\}$)
pSamples ← [sampleLLM(pPrompt) $* n_s$]
nSamples ← [sampleLLM(nPrompt) $* n_s$]
pScores ← sum(pSamples) / $n_s$
nScores ← sum(nSamples) / $n_s$
return pScores, nScores
```

CHAIN-OF-THOUGHT PROMPTING: A crucial component of getting interpretable and reliable likelihood estimates is to encourage the LLM to *justify* its choice by chain-of-thought prompting. As demonstrated in prior works, CoT elicits interpretability and reasoning capabilities in LLMs, and while we don't explicitly use the generated reasonings in our approach (great future work direction), we find that CoT improves the quality and

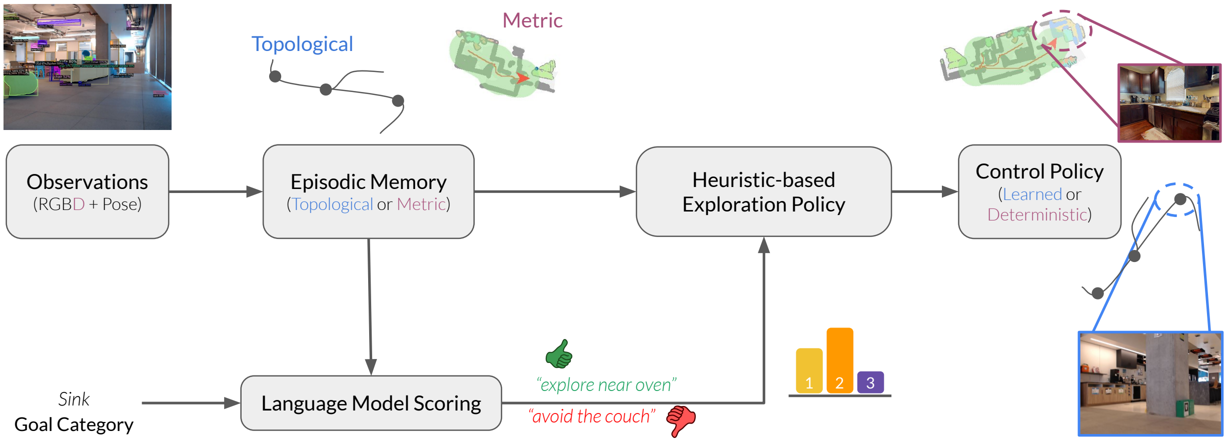


Figure 57: Overview of LFG for language-guided exploration. Based on the pose and observations, LFG builds an episodic memory (topological or metric), which is used by the heuristic-based exploration policy to rank adjacent clusters, or subgoal frontiers. Navigation to the subgoal frontier is completed by a low-level policy.

consistency of the likelihood estimates. Additionally, it also helps maintain interpretability, to better understand why the LFG-equipped agent takes certain decisions.

In summary, we score subgoals by sampling the LLM multiple times and empirically estimating the likelihood of each subgoal. We use a combination of positive and negative prompts to get unbiased likelihood estimates, and use chain-of-thought prompting to improve the quality and interpretability of the scores (Figure 56). We will now discuss how these scores can be incorporated into a navigation system as *search heuristics*.

10.5 LLM HEURISTICS FOR GOAL-DIRECTED EXPLORATION

Given the LLM scoring pipeline outlined in the previous section, our key insight is that we can incorporate these scores in a search-based planning pipeline to *heuristically* guide the search process. We instantiate LFG using frontier-based exploration (FBE) and LLM scores generated via polling.

FBE: This method maintains a "map" of the seen parts of the environment, which may be geometric [81] or topological [232], and a frontier separating it from the unexplored parts. By navigating to the *nearest* point of the frontier, the robot explores new areas of the environment until it finds the goal object or completes exploration without finding it. A standard FBE implementation is presented in Algorithm 10 inblack text. The robot maintains either a 2D metric map of its surroundings, or a topological map whose nodes are comprised of the robot's visual observations and edges represent paths taken in the environment. Additionally, we also store semantic labels corresponding to objects detected in the robot's observations, which are used to ground the observations in text.

At a fixed re-planning rate, the high-level planner computes its frontier f_i (Line 12), and picks the frontier point that is *closest* to the current location, i.e., maximizing the distance score (Line 18), and then navigates to this frontier (Line 26). At any point in this process, if the agent's semantic detector detects an object of the same category as the query q, it navigates directly to this object and the trajectory ends.

INCORPORATING LLM SCORES: Our method, LFG, extends FBE by using an additional search heuristic obtained by polling LLMs for semantic "scores". The modifications to FBE are marked in purple in Algorithm 10. After enumerating the frontiers, LFG uses the semantic labels from a VLM [301] to *ground* subgoal images at each frontier f_i (Line 13). These images are converted into textual strings, and form the subgoal candidates s_i that can be scored using Algorithm 9. We associate each frontier point f_i with the nearest object cluster c_i (Line 19), and compute LLM scores for each point as follows:

$$h(f_i, q) = w_p \cdot \mathrm{LLM}_{\mathrm{pos}}(c_i) - w_n \cdot \mathrm{LLM}_{\mathrm{neg}}(c_i) - \mathrm{dist}(f_i, p), \qquad (15)$$

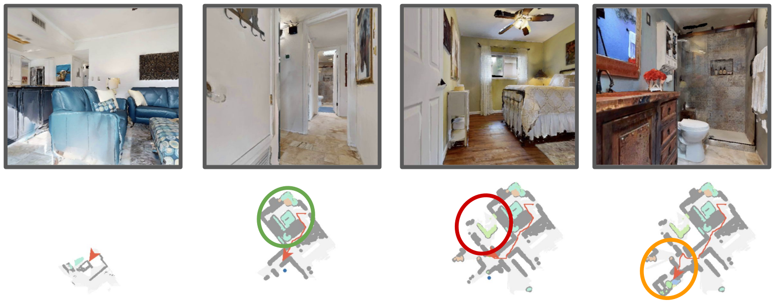
where p is the current position of the agent, and w_p, w_n are hyperparameters (see Appendix F.1.1). We then choose the frontier with the highest score to be the next subgoal (Line 26), navigate to it using a local controller, and repeat the planning process. Algorithm 10 outlines the general recipe for integrating LLM scores as a planning heuristic. Please see Appendix F.1 for specific instantiations of this system with geometric and topological maps, and more details about the referenced subroutines.

10.6 SYSTEM EVALUATION

We now evaluate the performance of LFG for the task of goal-directed exploration in real-world environments, and benchmark its performance against baselines. We instantiate two systems with LFG: a real-world system that uses a topological map and a learned control policy, and a simulated agent that uses a geometric map and a deterministic control policy. Our experiments show that both these systems outperform existing LLM-based exploration algorithms by a wide margin, owing to the high quality scores incorporated as search heuristics.

10.6.1 *Benchmarking ObjectNav Performance*

We benchmark the performance of LFG for the task of object-goal navigation on the Habitat ObjectNav Challenge [288], where the agent is placed into a simulated environment with photo-realistic graphics, and is tasked with finding a query object from one of 10 categories (e.g., "toilet", "bed", "couch" etc.). The simulated agent has access to egocentric RGBD observations and accurate pose information. We run 10 evaluation



Query: Find the potted plant.

LLM: "plants are not typically found in bedrooms or around furniture, so we should avoid cluster 1, 2, and 3"

Agent succeeds!

Figure 58: Qualitative example of a negative score influencing the agent's decision. LFG discourages the agent from exploring the bedroom and living room, leading to fast convergence toward the goal, whereas FBE fails. The CoT reasoning given by the LLM is shown in purple, justifying its score.

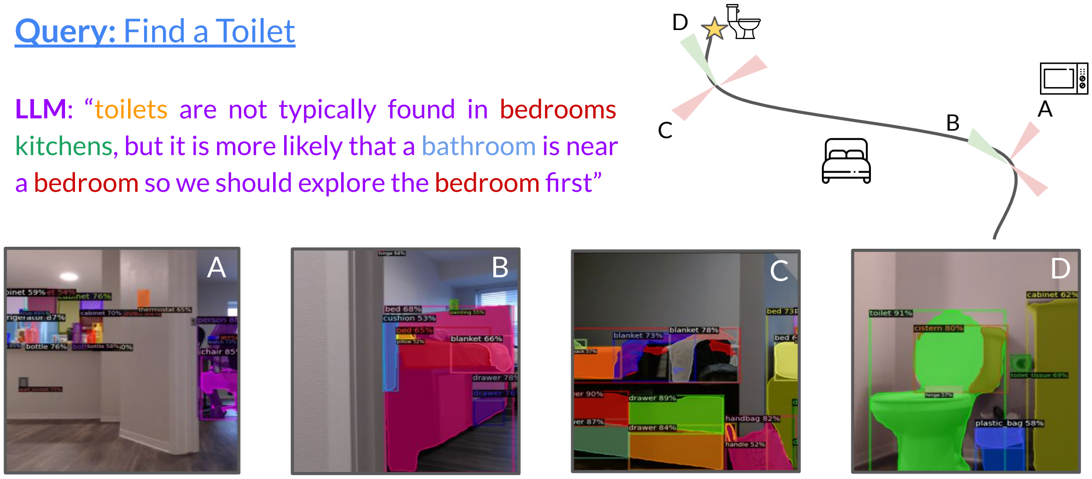
episodes per scene and report two metrics: the average success rate, and success weighted by optimal path length (SPL), the default metrics for the benchmark. Since LFG requires no training, we do not use the training scenes from HM3D.

We compare to three classes of published baselines: (i) learning-based baselines that learn navigation behavior from demonstrations or online experience in the training scenes [283] on up to 2.5B frames of experience, (ii) search-based baselines [29, 81], and (iii) LLM-based baselines that do not use the training data directly, but leverage the semantic knowledge inside foundation models to guide embodied tasks [296, 59].

Evaluating LFG on the HM3D benchmark, we find that it significantly outperforms search and LLM-based baselines (Table 11). Greedy LLM struggles on the task due to several LLM planning failures, causing the episodes to fail. LFG significantly outperforms the vanilla FBE baseline by leveraging semantic priors from LLMs to score subgoals intelligently. Comparing to learning-based baselines, we find that LFG outperforms most of them and closely matches the state-of-the-art on the task, proving the competence of our polling and heuristic approach. Figure 58 shows an example of the LFG agent successfully reaching the goal by using chain-of-thought and negative prompting.

L3MVN [296], which uses a combination of LLMs and search, performs slightly better than FBE, but is unable to fully leverage the semantics in LLMs. While being similar to LFG, it suffers from two key limitations: (i) it uses a small language model (GPT-2), which likely does not contain strong semantic priors for the agent to leverage, and (ii) it uses a simple likelihood-based scoring scheme, which we show below is not very effective. Another closely related work, LGX [59], uses a variant of greedy LLM scoring, and hence fails to perform reliably on the benchmark.

Probing deeper into the strong performance of LFG, we ablated various components of our scoring pipeline and studied the change in performance. Note that LGX (Greedy) and L3MVN (No CoT, Logprobs) can be seen as ablations of LFG. Table 12 shows that modifying both the prompting and scoring mechanisms used by LFG lead to large



Query: Find a Toilet

LLM: "toilets are not typically found in bedrooms kitchens, but it is more likely that a bathroom is near a bedroom so we should explore the bedroom first"

Figure 59: Qualitative example of LFG in real. LFG reasons about floor plans in the environment it is searching. In this apartment experiment, LFG believes that a bathroom is more likely to be found near a bedroom rather than a kitchen, and guides the robot towards the bedroom, successfully reaching the goal.

drops in performance. Most notably, scoring via polling (+7.8%) and CoT (+6.6%) are both essential to the strong performance of LFG. Furthermore, we find that using only positive prompts also hurts performance (−4.7%). Popular approaches for using LLMs for planning are significantly outperformed by LFG: Greedy (−14.5%) and Logprobs (−8.5%). Figure 58 shows an example of the LFG agent successfully reaching the goal by using CoT and negative prompting.

SETUP: For these experiments, we mimic the semantic mapping pipeline of the best-performing baseline on the benchmark [29, 81], and integrate LFG with the geometric map. The simulated agent builds a 2D semantic map of its environment, where grid cells represent both occupancy and semantic labels corresponding to objects detected by the agent. Prior work has shown that state-of-the-art vision models, such as DETIC, work poorly in simulation due to rendering artifacts [81]; hence, we use ground-truth semantic information for all simulated baselines to analyze navigation performance under perfect perception.

10.6.2 *Real-world Exploration with LFG*

To show the versatility of the LFG scoring framework, we further integrated it with a heuristic-based exploration framework that uses topological graphs for episodic memory [232]. We compare two published baselines: a language-agnostic FBE baseline [238], and an LLM-based baseline that uses the language model to greedily pick the frontier [59].

Method	Success	SPL	Data
DD-PPO [283]	27.9	14.2	2.5B
FBE [81]	61.1	34.0	0
SemExp [29]	63.1	0.29	10M
OVRL-v2 [290]	64.7	28.1	12M
Greedy LLM [59]	54.4	26.9	**0**
L3MVN [296]	62.4		**0**
LFG (Ours)	**68.9**	**36.0**	**0**

Table 11: LFG outperforms all LLM-based baselines on HM3D ObjectNav benchmark, and can achieve close to SOTA performance without any pre-training.

Method	Success	Δ
LFG (Full)	**68.9**	–
Prompting		
No CoT	62.3	(6.6)
Only Positives	64.2	(4.7)
Scoring		
Random	61.1	(7.8)
Logprobs	60.4	(8.5)
Ask	62.4	(6.5)

Table 12: We find that CoT prompting with positives and negatives, combined with polling, are essential to achieve the best performance.

We evaluate this system in two challenging real-world environments: a cluttered cafeteria and an apartment building (shown in Figures 57 and 59). In each environment, the robot is tasked to reach an object described by a textual string (e.g., "kitchen sink" or "oven"), and we measure the success rate and efficiency of reaching the goal. Episodes that take longer than 30 minutes are marked as failure. While we only tested our system with goal strings corresponding to the 20,000 classes supported by our object detector [301], this can be extended to more flexible goal specifications with the rapid progress in vision-language models.

We find that the complexity of real-world environments causes the language-agnostic FBE baseline to *time out*, i.e., the robot is unable to reach the goal by randomly exploring the environment. Both LLM baselines are able to leverage the stored semantic knowledge to guide the exploration in novel environments, but LFG achieves 16% better performance. Figure 59 shows an example rollout in a real apartment, where the robot uses LFG to reach the toilet successfully.

SETUP: We instantiate LFG in the real-world using a previously published topological navigation framework [232] that builds a topological map of the environment, where nodes correspond to the robot's visual observations and edges correspond to paths traversed in the environment. This system relies on omnidirectional RGB observations and does not attempt to make a dense geometric map of the environment. To obtain "semantic frontiers" from the omnidirectional camera, we generate $n_v = 4$ *views* and run an off-the-shelf object detector [301] to generate rich semantic labels describing objects in these views. The robot maintains a topological graph of these views and semantic labels, and picks the frontier view with the highest score (Algorithm 10, Line 26) according to LFG. The robot then uses a Transformer-based policy [242, 254] to reach this subgoal. For more implementation details, see Appendix F.1.3.

10.7 DISCUSSION

We presented LFG, a method for utilizing language models for *semantic guesswork* to help navigate to goals in new and unfamiliar environments. The central idea in our work is that, while language models can bring to bear rich semantic understanding, their ungrounded inferences about how to perform navigational tasks are better used as suggestions and heuristics rather than plans. We formulate a way to derive a heuristic score from language models that we can then incorporate into a planning algorithm, and use this heuristic planner to reach goals in new environments more effectively. This way of using language models benefits from their inferences when they are correct, and reverts to a more conventional unguided search when they are not.

LIMITATIONS AND FUTURE WORK While our experiments provide a validation of our key hypothesis, they have a number of limitations. First, we only test in indoor environments in both sim and real yet the role of semantics in navigation likely differs drastically across domains – e.g., navigating a forest might implicate semantics very differently than navigating an apartment building. Exploring the applicability of semantics derived from language models in other settings would be another promising and exciting direction for future work. Second, we acknowledge that multiple requests to cloud-hosted LLMs with CoT is slow and requires an internet connection, severely limiting the extent of real-world deployment of the proposed method. We hope that ongoing advancements in quantizing LLMs for edge deployment and fast inference will address this limitation.

ACKNOWLEDGMENTS

This research was partly supported by AFOSR FA9550-22-1-0273 and DARPA ANSR. The authors would like to thank Bangguo Yu, Vishnu Sashank Dorbala, Mukul Khanna, Theophile Gervet, and Chris Paxton, for their help in reproducing baselines. The authors would also like to thank Ajay Sridhar for supporting real-world experiments, and Devendra Singh Chaplot, Jie Tan, Peng Xu, and Tingnan Zhang, for useful discussions in various stages of the project.

Algorithm 10 Instantiating LFG for Goal-Directed Exploration

```
Input: $o_0$, Goal language query $q$
subgoal ← None
while not done do
    $o_t$ ← getObservation()
    episodicMemory ← mappingModule($o_t$)
    if $q$ in semanticMap then
        subGoal ← getLocation(episodicMemory, q)
    else
        if  numSteps % $\tau == 0$ then
            // replanning
            location ← getCurrentLocation()
            frontier ← getFrontier(episodicMemory)
            objectClusters ← getSemanticLabels(episodicMemory, frontier)
            $LLM_{pos}, LLM_{neg}$ ← ScoreSubgoals(objectClusters)
            scores ← []
            for point in frontier do
                distance ← DistTo(location, point)
                scores[point] ← - distance
                closestCluster ← getClosestCluster(objectClusters, point)
                $i$ ← clusterID(closestCluster)
                if dist(closestCluster, point) $< \delta$ then
                    // incorporate language scores
                    scores[point] ← $w_p \cdot LLM_{pos}[i] - w_n \cdot LLM_{neg}[i] - distance$
                end if
                end for
                subgoal ← argmax(scores)
            end if
        end if
        numSteps ← numSteps $+1$
        goTo(subGoal)
    end while
```

11

CONCLUSION

In this dissertation, we presented a recipe for deploying intelligent robots in open-world environments by training cross-embodiment robot foundation models, and combining them with internet foundation models of language and vision. We explored the design space of datasets, algorithms and architectures for large-scale robot learning (Chapters 2–5), proposed the idea of cross-embodiment policy learning and open-sourced three increasingly capable variants of navigation foundation models (Chapters 6–8), and proposed a novel planning based framework that can combine these RFMs with pretrained vision and language models to follow natural language instructions in real-world environments (Chapters 9–10).

We are particularly excited about the role of plug-and-play robot foundation models (Definition 1) in reducing the entry barrier to robotics and embodied AI for both researchers and practitioners, and in addressing the reproducibility crisis in robot learning. Broadly capable robot models that can be deployed in any environment and on any robot will enable ever increasing number of downstream of applications, and improve reliability of empirical evaluations in real-world experiments. Robot foundation models open-sourced as a part of this dissertation have already been deployed *in-the-wild* on 25+ robotic systems around the world, ranging from campus delivery robots in Finland [258], to an autonomous orienteering robot in Estonia[1], drones in Princeton [247], and an industrial autonomous overhead crane[2].

While the specific formulation of cross-embodiment learning in this dissertation is limited to visual navigation, there is a lot of promise in extending this idea to a broader set of tasks. Preliminary investigations in robotic manipulation, performed as a part of the Open X-Embodiment Collaboration, indeed show that formulating the problem in a similar way — by homogenizing the observation and action spaces, using an embodiment prompt, and training on heterogeneous cross-embodiment datasets — indeed enables broad generalization and positive transfer across various robotic embodiments, such as a WidowX robot in Berkeley, a Franka Emika Panda in Germany, and Hello Robot Stretch in New York City [195]. We also see similar positive signals when exploring more complex problems like egocentric manipulation and mobile manipulation, where a combination of clever architectures and aligning action spaces across robots enables generalization across

[1] Autonomous Driving Lab @ University of Tartu

[2] Konecranes

mobility and manipulation, and opens doors to training a single robot policy across a broader set of robots [293]. Despite these signals, it must be noted that assumptions like homogenized action spaces tend to hurt dexterous tasks a lot more than the simpler navigation tasks, since a lot of embodiment-specific information is abstracted away from the model. However, we remain optimistic of the potential of sharing data across robotic embodiments, and hope that future research will help bridge this gap to more dexterous and fine-grained control.

Finally, we acknowledge that training highly capable pre-trained backbones is an important, but small, piece in the larger puzzle of building robots that are ready for open-world deployment. Deployment of intelligent robots in challenging, unstructured environments will require critical breakthroughs in adjacent areas of robotics and machine learning, such as robust fine-grained manipulation hardware, continual learning, reinforcement learning, efficient edge computing, flexible neural network architectures, generative modeling, safety, fast online adaptation, and much more.

OPEN PROBLEMS AND FUTURE AVENUES

We foresee a number of exciting research directions that can be enabled by the research presented in this dissertation:

Scaling up cross-embodiment learning. Building on the presented work on cross-embodiment learning in navigation [234, 242], we anticipate exploration of its merits as *a general framework for learning-based grasping, locomotion, and mobile manipulation*. In a recent collaboration, we found that cross-embodiment training can indeed help grasping [195], but we are yet to effectively use the large amount of robot data we have access to. This presents a unique opportunity in machine learning, where data is typically expensive, and we plan to design architectures and algorithms that can learn useful behaviors from large-scale data. It would also be exciting to explore applications of these generalizable policies to scenarios where robots do not have access to high-quality sensors beyond RGB cameras (e.g., underwater exploration, miniature robots), and with limited potential for simulation or data collection (e.g., drones with limited battery life, robot-assisted vascular surgery, search-and-rescue in forests and other dense unstructured environments).

Safety and robustness. Robots deployed in physical spaces must exhibit robustness to environmental perturbations and avoid unsafe maneuvers, that can lead to catastrophic consequences. Our work (Chapter 6) has shown that generalization of cross-embodiment policies leads to *emergent* robustness to natural and adversarial factors of perturbation, such as varying lighting conditions, rain or damaged hardware. While these learned policies were empirically robust, they are not guaranteed to be safe. Previous work [51] has developed a suitable solution lies in combining empirically robust policies with a "fallback" verifiable planner, which could be an interesting avenue to explore. Another mechanism is to have safety constraints that can be incorporated during deployment using sample-efficient online RL, which has shown great promise for autonomous improvement in recent work [233]. This can enable exploring self-supervised reward signals for out-of-distribution detection and automatically learning recovery maneuvers from suboptimal data.

How might robotics help language, vision, and generative model research? The rise of large generative models of language and vision has enabled a large number of useful applications that leverage these *foundation* models, trained on internet-scale text and images, to provide a simplistic version of "commonsense reasoning". However, these models are not *grounded* in the physical world and tend to hallucinate missing details. Our preliminary investigations suggest that language models can be grounded by combining them with skill-based affordances [240, 108], treating robots as a *consumer* of knowledge stored in these models. A further exciting direction could be the ability to *jointly* train foundation models of language, vision, and robot control, which can be trained on internet-scale data and simultaneously, be grounded in the physical world. Cognitive scientists believe that embodied intelligence in humans emerges as a result of sensorimotor activity (the embodiment hypothesis [273]). Its an exciting time to test this hypothesis for artificial intelligence using the abundance of cross-embodiment data, combined with internet-scale text, images and videos. We believe that data from real-world interactions of robots and control-based training objectives can lead to improved grounded reasoning in large language and vision models, and will design algorithms and systems that can leverage large-scale robot data.

How might robots co-exist with humans? In the future, robots will not exist in instrumented, static environments with a well-defined task, but co-habitate spaces with humans. They need to understand how humans interact with them, how humans are influenced by them, and how humans can help robots to cooperatively accomplish tasks such as cleaning a house, preparing a meal, or stocking inventory in a hospital. This poses two interesting challenges that I am excited to explore: (i) *How do we model the rich interactions between humans and robots?*, and (ii) *How do we incorporate human feedback into the autonomous learning process, so robots can be better assistants and companions?* Understanding these questions will be imperative to the deployment of autonomous robots in open-world environments alongside humans.

BIBLIOGRAPHY

[1] Alessandro Achille and Stefano Soatto. “Emergence of invariance and disentanglement in deep representations”. In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 1947–1980.

[2] Evan Ackerman. “Skydio demonstrates incredible obstacle-dodging full autonomy with new R1 consumer drone”. In: *IEEE Spectrum* (2018).

[3] Ali Agha et al. *NeBula: Quest for Robotic Autonomy in Challenging Environments; TEAM CoSTAR at the DARPA Subterranean Challenge*. 2021.

[4] Michael Ahn et al. “Do As I Can, Not As I Say: Grounding Language in Robotic Affordances”. In: *arXiv preprint arXiv:2204.01691*. 2022.

[5] Jean-Baptiste Alayrac et al. *Flamingo: a Visual Language Model for Few-Shot Learning*. 2022.

[6] Alexander A Alemi et al. “Deep variational information bottleneck”. In: *arXiv preprint arXiv:1612.00410* (2016).

[7] Peter Anderson et al. *On Evaluation of Embodied Navigation Agents*. 2018. arXiv: 1807.06757 [cs.AI].

[8] Peter Anderson et al. “Sim-to-Real Transfer for Vision-and-Language Navigation”. In: *Proceedings of the 2020 Conference on Robot Learning*. 2021.

[9] Peter Anderson et al. “Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3674–3683.

[10] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. 2017.

[11] Mikel Artetxe et al. “Efficient large scale language modeling with mixtures of experts”. In: *arXiv preprint arXiv:2112.10684* (2021).

[12] Ruzena Bajcsy, Yiannis Aloimonos, and John K Tsotsos. “Revisiting active perception”. In: *Autonomous Robots* (2018).

[13] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. “ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst”. In: *Robotics: Science and Systems (RSS)*. 2019.

[14] Marc G Bellemare et al. “Unifying count-based exploration and intrinsic motivation”. In: *arXiv preprint arXiv:1606.01868* (2016).

[15] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *ArXiv* (2013).

[16] Joydeep Biswas and Manuela M. Veloso. "Localization and navigation of the CoBots over long-term deployments". In: *The International Journal of Robotics Research* (2013).

[17] Sid Black et al. *GPT-NeoX-20B: An Open-Source Autoregressive Language Model*. 2022.

[18] Valts Blukis et al. "Following High-level Navigation Instructions on a Simulated Quadcopter with Imitation Learning". In: *CoRR* (2018).

[19] Johann Borenstein and Yoram Koren. "Real-time obstacle avoidance for fast mobile robots". In: *IEEE Transactions on systems, Man, and Cybernetics* (1989).

[20] F. Bourgault et al. "Information based adaptive robotic exploration". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2002.

[21] Anthony Brohan et al. "RT-1: Robotics Transformer for Real-World Control at Scale". In: *arXiv preprint* (2023).

[22] Tom Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. 2020.

[23] Jake Bruce et al. "Learning Deployable Navigation Policies at Kilometer Scale from a Single Traversal". In: *Conference on Robot Learning (CoRL)*. 2018.

[24] Yuri Burda et al. "Exploration by random network distillation". In: *arXiv preprint arXiv:1810.12894* (2018).

[25] Nicholas Carlevaris-Bianco, Arash K Ushani, and Ryan M Eustice. "University of Michigan North Campus long-term vision and lidar dataset". In: *The International Journal of Robotics Research* (2016).

[26] Joao Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[27] Matthew Chang, Arjun Gupta, and Saurabh Gupta. "Semantic Visual Navigation by Watching Youtube Videos". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[28] Devendra Singh Chaplot et al. "Learning to Explore using Active Neural SLAM". In: *International Conference on Learning Representations (ICLR)*. 2020.

[29] Devendra Singh Chaplot et al. "Semantic Curiosity for Active Visual Learning". In: *ECCV*. 2020.

[30] Benjamin Charrow et al. "Information-theoretic mapping using cauchy-schwarz quadratic mutual information". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[31] Yevgen Chebotar et al. “Q-Transformer: Scalable Offline Reinforcement Learning via Autoregressive Q-Functions”. In: *7th Annual Conference on Robot Learning (CoRL)*. 2023.

[32] Yevgen Chebotar et al. “Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills”. In: *arXiv preprint arXiv:2104.07749* (2021).

[33] Boyuan Chen et al. “Open-vocabulary queryable scene representations for real world planning”. In: *arXiv preprint arXiv:2209.09874* (2022).

[34] Chenyi Chen et al. “Deepdriving: Learning affordance for direct perception in autonomous driving”. In: *IEEE International Conference on Computer Vision*. 2015.

[35] David L. Chen and Raymond J. Mooney. “Learning to Interpret Natural Language Navigation Instructions from Observations”. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.

[36] Howard Chen et al. “TOUCHDOWN: Natural Language Navigation and Spatial Reasoning in Visual Street Environments”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[37] Lili Chen et al. “Decision Transformer: Reinforcement Learning via Sequence Modeling”. In: *Advances in Neural Information Processing Systems*. 2021.

[38] Mark Chen et al. “Evaluating Large Language Models Trained on Code”. In: *arXiv* (2021).

[39] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In: *International Conference on Machine Learning (ICML)* (2020).

[40] Xi Chen et al. “Pali: A jointly-scaled multilingual language-image model”. In: *arXiv preprint arXiv:2209.06794* (2022).

[41] Yen-Chun Chen et al. “Uniter: Universal image-text representation learning”. In: *ECCV*. 2020.

[42] Cheng Chi et al. “Diffusion Policy: Visuomotor Policy Learning via Action Diffusion”. In: *Robotics: Science and Systems (RSS)*. 2023.

[43] Hao-Tien Lewis Chiang et al. “Learning Navigation Behaviors End-to-End with AutoRL”. In: *IEEE Robotics and Automation Letters* (2019).

[44] F. Codevilla et al. “End-to-End Driving Via Conditional Imitation Learning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[45] F. Codevilla et al. “End-to-End Driving Via Conditional Imitation Learning”. In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[46] Cédric Colas et al. “CURIOUS: intrinsically motivated modular multi-goal reinforcement learning”. In: *International conference on machine learning*. PMLR. 2019.

[47] Robert Dadashi et al. “Continuous Control with Action Quantization from Demonstrations”. In: *39th International Conference on Machine Learning (ICML)*. 2022.

[48] FR Dalgleish, SW Tetlow, and RL Allwood. "Vision-based navigation of unmanned underwater vehicles: a survey. Part I: Vision Based Cable-, Pipeline-and Fish Tracking". In: *Journal of Marine Design and Operations*. 7. 2004, pp. 51–56.

[49] DARPA. *Subterranean Challenge*. 2019. URL: https://www.subtchallenge.com.

[50] Sudeep Dasari, Frederik Ebert, et al. "RoboNet: Large-Scale Multi-Robot Learning". In: *Conference on Robot Learning (CoRL)*. 2020.

[51] Nitish Dashora et al. "Hybrid Imitative Planning with Geometric and Predictive Costs in Off-road Environments". In: *International Conference on Robotics and Automation (ICRA)*. 2022.

[52] Andrew J Davison and David W Murray. "Mobile robot localisation using active vision". In: *European conference on computer vision*. Springer. 1998, pp. 809–825.

[53] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[54] Coline Devin et al. "Learning modular neural network policies for multi-task and multi-robot transfer". In: *2017 International Conference on Robotics and Automation (ICRA)*. 2017.

[55] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* (1959).

[56] Yan Ding et al. *Task and Motion Planning with Large Language Models for Object Rearrangement*. 2023.

[57] Yiming Ding et al. "Goal-conditioned Imitation Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[58] Yiming Ding et al. "Goal-conditioned imitation learning". In: *arXiv preprint arXiv:1906.05838* (2019).

[59] Vishnu Sashank Dorbala, James F. Jr. Mullen, and Dinesh Manocha. *Can an Embodied Agent Find Your "Cat-shaped Mug"? LLM-Based Zero-Shot Object Navigation*. 2023.

[60] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*. 2021.

[61] Danny Driess et al. "PaLM-E: An Embodied Multimodal Language Model". In: *arXiv preprint arXiv:2303.03378*. 2023.

[62] Yan Duan et al. "RL^2: Fast reinforcement learning via slow reinforcement learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[63] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. "Challenges of real-world reinforcement learning". In: *arXiv preprint arXiv:1904.12901* (2019).

[64] Lasse Espeholt et al. “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures”. In: *ICML*. 2018.

[65] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. “Search on the Replay Buffer: Bridging Planning and RL”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[66] Ben Eysenbach et al. “Rewriting history with inverse rl: Hindsight inference for policy improvement”. In: *Advances in Neural Information Processing Systems* (2020).

[67] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. “C-Learning: Learning to Achieve Goals via Recursive Classification”. In: *arXiv preprint arXiv:2011.08909* (2020).

[68] Kuan Fang et al. “Generalization with Lossy Affordances: Leveraging Broad Offline Data for Learning Visuomotor Tasks”. In: *6th Annual Conference on Robot Learning*. 2022.

[69] A. Faust et al. “PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[70] Aleksandra Faust et al. “PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018.

[71] Gilbert Feng et al. “GenLoco: Generalized Locomotion Controllers for Quadrupedal Robots”. In: *Conference on Robot Learning (CoRL)*. 2022.

[72] Pete Florence et al. “Implicit Behavioral Cloning”. In: *5th Annual Conference on Robot Learning (CoRL)*. 2021.

[73] Patrick S. Foo et al. “Do humans integrate routes into a cognitive map? Map-versus landmark-based navigation of novel shortcuts.” In: *Journal of experimental psychology. Learning, memory, and cognition* (2005).

[74] Patrick S. Foo et al. “Do humans integrate routes into a cognitive map? Map-versus landmark-based navigation of novel shortcuts.” In: *Journal of experimental psychology. Learning, memory, and cognition* (2005).

[75] A. Francis et al. “Long-Range Indoor Navigation With PRM-RL”. In: *IEEE Transactions on Robotics* (2020).

[76] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Manuel Rendón-Mancha. “Visual simultaneous localization and mapping: a survey”. In: *Artificial Intelligence Review* (2015).

[77] Scott Fujimoto, David Meger, and Doina Precup. “Off-policy deep reinforcement learning without exploration”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2052–2062.

[78] Paul Furgale and Timothy D Barfoot. “Visual teach and repeat for long-range rover autonomy”. In: *Journal of Field Robotics* (2010).

[79] Shani Gamrian and Yoav Goldberg. “Transfer learning for related reinforcement learning tasks via image-to-image translation”. In: *International Conference on Machine Learning*. PMLR. 2019.

[80] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. “Learning to fly by crashing”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[81] Theophile Gervet et al. “Navigating to objects in the real world”. In: *Science Robotics* (2023).

[82] Dibya Ghosh et al. “Learning to reach goals without reinforcement learning”. In: *arXiv preprint arXiv:1912.06088* (2019).

[83] S. Gillner and H. A. Mallot. “Navigation and Acquisition of Spatial Knowledge in a Virtual Maze”. In: *Journal of Cognitive Neuroscience* (1998).

[84] Anirudh Goyal et al. “Infobot: Transfer and exploration via the information bottleneck”. In: *arXiv preprint arXiv:1901.10902* (2019).

[85] Kristen Grauman et al. “Ego4D: Around the World in 3,000 Hours of Egocentric Video”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[86] Jing Gu et al. “Vision-and-Language Navigation: A Survey of Tasks, Methods, and Future Directions”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022.

[87] Xiuye Gu et al. “Open-vocabulary Object Detection via Vision and Language Knowledge Distillation”. In: *International Conference on Learning Representations*. 2022.

[88] Abhishek Gupta et al. “Learning invariant feature spaces to transfer skills with reinforcement learning”. In: *arXiv preprint arXiv:1703.02949* (2017).

[89] Meera Hahn et al. “No RL, No Simulation: Learning to Navigate without Navigating”. In: *Neural Information Processing Systems (NeurIPS*. 2021.

[90] Philippe Hansen-Estruch et al. *IDQL: Implicit Q-Learning as an Actor-Critic Method with Diffusion Policies*. 2023.

[91] Kristian Hartikainen et al. “Dynamical Distance Learning for Semi-Supervised and Unsupervised Skill Discovery”. In: *International Conference on Learning Representations*. 2020.

[92] Elad Hazan et al. “Provably efficient maximum entropy exploration”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2681–2691.

[93] Kaiming He et al. “Mask R-CNN”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017.

[94] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[95] Karl Moritz Hermann et al. “Learning To Follow Directions in Street View”. In: *CoRR* (2019).

[96] Noriaki Hirose et al. “Deep visual MPC-policy learning for navigation”. In: *IEEE Robotics and Automation Letters* (2019).

[97] Noriaki Hirose et al. “ExAug: Robot-Conditioned Navigation Policies via Geometric Experience Augmentation”. In: *International Conference on Robotics and Automation (ICRA)*. 2023.

[98] Noriaki Hirose et al. “SACSoN: Scalable Autonomous Control for Social Navigation”. In: *arXiv preprint arXiv:2306.01874* (2023).

[99] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising Diffusion Probabilistic Models”. In: *Neural Information Processing Systems*. 2020.

[100] Jonathan Ho and Tim Salimans. “Classifier-free diffusion guidance”. In: *arXiv preprint arXiv:2207.12598* (2022).

[101] Dirk Holz et al. *A comparative evaluation of exploration strategies and heuristics to improve them*. 2011.

[102] Matthew Honnibal et al. “spaCy: Industrial-strength natural language processing in python”. In: (2020).

[103] Rein Houthooft et al. “Vime: Variational information maximizing exploration”. In: *arXiv preprint arXiv:1605.09674* (2016).

[104] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[105] Chenguang Huang et al. “Visual Language Maps for Robot Navigation”. In: *arXiv preprint arXiv:2210.05714* (2022).

[106] Shaohan Huang et al. *Language Is Not All You Need: Aligning Perception with Language Models*. 2023.

[107] Wenlong Huang, Igor Mordatch, and Deepak Pathak. “One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control”. In: *International Conference on Machine Learning (ICML)*. 2020.

[108] Wenlong Huang et al. *Grounded Decoding: Guiding Text Generation with Grounded Models for Robot Control*. 2023.

[109] Wenlong Huang et al. “Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents”. In: *arXiv preprint arXiv:2201.07207* (2022).

[110] Wenlong Huang et al. "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents". In: *International Conference on Machine Learning (ICML)*. 2022.

[111] Brian Ichter et al. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *Annual Conference on Robot Learning (CoRL)*. 2022.

[112] Maximilian Igl et al. "Generalization in reinforcement learning with selective noise injection and information bottleneck". In: *arXiv preprint arXiv:1910.12911* (2019).

[113] Vihan Jain et al. "Stay on the Path: Instruction Fidelity in Vision-and-Language Navigation". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019.

[114] Eric Jang et al. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: *5th Annual Conference on Robot Learning*. 2021.

[115] Michael Janner et al. "Planning with Diffusion for Flexible Behavior Synthesis". In: *International Conference on Machine Learning (ICML)*. 2022.

[116] Krishna Murthy Jatavallabhula, Ganesh Iyer, and Liam Paull. "gradSLAM: Dense SLAM meets Automatic Differentiation". In: *CoRR* (2019).

[117] Krishna Murthy Jatavallabhula et al. "ConceptFusion: Open-set Multimodal 3D Mapping". In: *arXiv* (2023).

[118] Chao Jia et al. "Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision". In: *Proceedings of the 38th International Conference on Machine Learning*. 2021.

[119] Yunfan Jiang et al. "VIMA: General Robot Manipulation with Multimodal Prompts". In: *arXiv preprint* (2022).

[120] Yunfan Jiang et al. "Vima: General robot manipulation with multimodal prompts". In: *arXiv preprint arXiv:2210.03094* (2023).

[121] Abhishek Kadian et al. "Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance?" In: *IEEE Robotics and Automation Letters* (2020).

[122] Leslie Pack Kaelbling. "Learning to achieve goals". In: *IJCAI*. Citeseer. 1993, pp. 1094–1099.

[123] G. Kahn et al. "Self-Supervised Deep RL with Generalized Computation Graphs for Robot Navigation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[124] Gregory Kahn, Pieter Abbeel, and Sergey Levine. "BADGR: An Autonomous Self-Supervised Learning-Based Navigation System". In: *IEEE Robotics and Automation Letters* (2021).

[125] Gregory Kahn et al. "Composable action-conditioned predictors: Flexible off-policy learning for robot navigation". In: *Conference on Robot Learning*. 2018.

[126] Dmitry Kalashnikov et al. “Mt-opt: Continuous multi-task robotic reinforcement learning at scale”. In: *arXiv preprint arXiv:2104.08212* (2021).

[127] Katie Kang, Gregory Kahn, and Sergey Levine. “Hierarchically Integrated Models: Learning to Navigate from Heterogeneous Robots”. In: *Conference on Robot Learning (CoRL)*. 2021.

[128] Jared Kaplan et al. “Scaling Laws for Neural Language Models”. In: *CoRR* (2020).

[129] Haresh Karnan et al. “Socially CompliAnt Navigation Dataset (SCAND): A Large-Scale Dataset Of Demonstrations For Social Navigation”. In: *IEEE Robotics and Automation Letters* (2022).

[130] Apoorv Khandelwal et al. “Simple but Effective: CLIP Embeddings for Embodied AI”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[131] Diederik Kingma et al. “Variational diffusion models”. In: *Neural Information Processing Systems (NeurIPS)* (2021).

[132] Diederik P Kingma and Max Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[133] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)* (2015).

[134] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2009.

[135] Thomas Kollar and Nicholas Roy. “Efficient Optimization of Information-Theoretic Exploration in SLAM”. In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 2008.

[136] Eric Kolve et al. “AI2-THOR: An Interactive 3D Environment for Visual AI”. In: *ArXiv* (2019).

[137] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. “Offline reinforcement learning with implicit q-learning”. In: *arXiv preprint arXiv:2110.06169* (2021).

[138] Ilya Kostrikov et al. “Offline reinforcement learning with fisher divergence critic regularization”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5774–5783.

[139] Alexandros Kouris and Christos-Savvas Bouganis. “Learning to Fly by MySelf: A Self-Supervised CNN-Based Approach for Autonomous Navigation”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.

[140] Jacob Krantz et al. “Beyond the Nav-Graph: Vision-and-Language Navigation in Continuous Environments”. In: *Proceedings of the European Conference on Computer Vision*. 2020.

[141] Eric Krotkov and Martial Hebert. “Mapping and positioning for a prototype lunar rover”. In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. IEEE. 1995.

[142] Alexander Ku et al. “Room-Across-Room: Multilingual Vision-and-Language Navigation with Dense Spatiotemporal Grounding”. In: *Conference on Empirical Methods for Natural Language Processing (EMNLP)*. 2020.

[143] Benjamin Kuipers and Yung-Tai Byun. “A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations”. In: *Robotics and Autonomous Systems* (1991). Special Issue Toward Learning Robots.

[144] Ashish Kumar et al. “Rma: Rapid motor adaptation for legged robots”. In: *Robotics: Science and Systems*. 2021.

[145] Aviral Kumar et al. “Conservative q-learning for offline reinforcement learning”. In: *Advances in Neural Information Processing Systems* (2020).

[146] Ashish Kumar* et al. “Visual Memory for Robust Path Following”. In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[147] T.L Lai and Herbert Robbins. “Asymptotically efficient adaptive allocation rules”. In: *Advances in Applied Mathematics* (1985).

[148] Sascha Lange, Thomas Gabel, and Martin Riedmiller. “Batch reinforcement learning”. In: *Reinforcement learning*. Springer, 2012.

[149] Steven M LaValle. *Planning Algorithms*. Cambridge university press, 2006.

[150] Alessandro Lazaric. “Transfer in reinforcement learning: a framework and a survey”. In: *Reinforcement Learning*. Springer, 2012.

[151] Alex X Lee et al. “Beyond pick-and-place: Tackling robotic stacking of diverse shapes”. In: *5th Annual Conference on Robot Learning*. 2021.

[152] Lisa Lee et al. “Efficient exploration via state marginal matching”. In: *arXiv preprint arXiv:1906.05274* (2019).

[153] Brian Lester, Rami Al-Rfou, and Noah Constant. “The Power of Scale for Parameter-Efficient Prompt Tuning”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2021.

[154] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *The Journal of Machine Learning Research* (2016).

[155] Sergey Levine et al. *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. 2020.

[156] Chengshu Li et al. “HRL4IN: Hierarchical Reinforcement Learning for Interactive Navigation with Mobile Manipulators”. In: *3rd Annual Conference on Robot Learning (CoRL)*. 2019.

[157] Liunian Harold Li et al. “VisualBERT: A Simple and Performant Baseline for Vision and Language”. In: *Arxiv*. 2019.

[158] Kevin Lin et al. *Text2Motion: From Natural Language Instructions to Feasible Plans*. 2023.

[159] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[160] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *ECCV*. 2014.

[161] Kara Liu et al. “Hallucinative topological memory for zero-shot visual planning”. In: *International Conference on Machine Learning*. PMLR. 2020.

[162] Xiaosen Liu et al. “The Power of Scale for Parameter-Efficient Prompt Tuning”. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2021.

[163] Antonio Loquercio et al. “DroNet: Learning to Fly by Driving”. In: *IEEE Robotics and Automation Letters* (2018).

[164] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[165] Corey Lynch et al. “Learning latent plans from play”. In: *Conference on Robot Learning*. PMLR. 2020.

[166] Arjun Majumdar et al. “Improving vision-and-language navigation with image-text pairs from the web”. In: *Proceedings of the European Conference on Computer Vision*. 2020.

[167] Arjun Majumdar et al. “Where are we in the search for an Artificial Visual Cortex for Embodied Intelligence?” In: *arXiv preprint arXiv:2303.18240* (2023).

[168] Arjun Majumdar et al. “Zson: Zero-shot object-goal navigation using multimodal goal embeddings”. In: *arXiv preprint arXiv:2206.12403* (2022).

[169] Travis Manderson et al. “Self-Supervised, Goal-Conditioned Policies for Navigation in Unstructured Environments”. In: 2010.

[170] Ajay Mandlekar et al. “Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020.

[171] Manolis Savva* et al. “Habitat: A Platform for Embodied AI Research”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[172] Cynthia Matuszek, Dieter Fox, and Karl Koscher. “Following directions using statistical machine translation”. In: *2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2010.

[173] Cynthia Matuszek et al. “Learning to Parse Natural Language Commands to a Robot Control System”. In: *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. 2013.

[174] John McCormac et al. “SemanticFusion: Dense 3D semantic mapping with convolutional neural networks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[175] Oier Mees, Jessica Borja-Diaz, and Wolfram Burgard. *Grounding Language with Visual Affordances over Unstructured Data*. 2023.

[176] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. "Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences". In: *AAAI*. 2016.

[177] M. Meng and A. C. Kak. "Mobile Robot Navigation using Neural Networks and Nonmetrical Environmental Models". In: *IEEE Control Systems Magazine* (1993).

[178] M. Meng and A. C. Kak. "NEURO-NAV: A Neural Network based Architecture for Vision-guided Mobile Robot Navigation using Non-metrical Models of the Environment". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 1993.

[179] X. Meng et al. "Scaling Local Control to Large-Scale Topological Navigation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[180] Luke Metz et al. *Discrete Sequential Prediction of Continuous Actions for Deep RL*. 2019.

[181] Atanas Mirchev et al. "Approximate bayesian inference in spatial environments". In: *arXiv preprint arXiv:1805.07206* (2018).

[182] Piotr Mirowski et al. "The StreetLearn Environment and Dataset". In: *CoRR* (2019).

[183] Dipendra Kumar Misra et al. "Mapping Instructions to Actions in 3D Environments with Visual Goal Prediction". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018.

[184] Anusha Nagabandi et al. "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning". In: *arXiv preprint arXiv:1803.11347* (2018).

[185] A. Nair et al. "Contextual Imagined Goals for Self-Supervised Robotic Learning". In: *Conference on Robot Learning (CoRL)*. 2019.

[186] Ashvin Nair et al. "Awac: Accelerating online reinforcement learning with offline datasets". In: *arXiv preprint arXiv:2006.09359* (2020).

[187] Suraj Nair et al. "R3M: A Universal Visual Representation for Robot Manipulation". In: *Conference on Robot Learning (CoRL)*. 2022.

[188] Medhini Narasimhan et al. "Seeing the Un-Scene: Learning Amodal Semantic Maps for Room Navigation". In: *CoRR* (2020).

[189] Soroush Nasiriany et al. "Planning with Goal-Conditioned Policies". In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[190] Nigamaa Nayakanti et al. *Wayformer: Motion Forecasting via Simple and Efficient Attention Networks*. 2022.

[191] Alexander Quinn Nichol and Prafulla Dhariwal. "Improved Denoising Diffusion Probabilistic Models". In: *International Conference on Machine Learning (ICML)*. 2021.

[192] Takahiro Niwa, Shun Taguchi, and Noriaki Hirose. "Spatio-Temporal Graph Localization Networks for Image-based Navigation". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2022.

[193] John O'Keefe and Lynn Nadel. *The Hippocampus as a Cognitive Map*. Oxford: Clarendon Press, 1978.

[194] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019.

[195] Open X-Embodiment Collaboration et al. "Open X-Embodiment: Robotic Learning Datasets & RT-X Models". In: *arXiv* (2023).

[196] Simone Parisi et al. "The Unsurprising Effectiveness of Pre-Trained Vision Models for Control". In: *arXiv preprint arXiv:2203.03580* (2022).

[197] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. "Actor-mimic: Deep multitask and transfer reinforcement learning". In: *arXiv preprint arXiv:1511.06342* (2015).

[198] Deepak Pathak et al. "Curiosity-driven exploration by self-supervised prediction". In: *International Conference on Machine Learning*. PMLR. 2017.

[199] Tim Pearce et al. "Imitating Human Behaviour with Diffusion Models". In: *The Eleventh International Conference on Learning Representations (ICLR)*. 2023.

[200] Xue Bin Peng et al. "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *International Conference on Robotics and Automation (ICRA)*. 2018.

[201] Silviu Pitis et al. "Maximum entropy gain exploration for long horizon multi-goal reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020.

[202] Patrick von Platen et al. *Diffusers: State-of-the-art diffusion models*. https://github.com/huggingface/diffusers. 2022.

[203] Vitchyr Pong et al. "Temporal difference models: Model-free deep rl for model-based control". In: *arXiv preprint arXiv:1802.09081* (2018).

[204] Vitchyr H Pong et al. "Skew-fit: State-covering self-supervised reinforcement learning". In: *arXiv preprint arXiv:1903.03698* (2019).

[205] Alec Radford et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning (ICML)*. 2021.

[206] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[207] Ilija Radosavovic et al. "Real-World Robot Learning with Masked Visual Pre-training". In: *Conference on Robot Learning (CoRL)*. 2022.

[208] Santhosh Kumar Ramakrishnan et al. "Poni: Potential functions for objectgoal navigation with interaction-free learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

[209] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022.

[210] Ali Sharif Razavian et al. "CNN Features off-the-shelf: an Astounding Baseline for Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[211] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems*. 2015.

[212] Moritz Reuss et al. "Goal Conditioned Imitation Learning using Score-based Diffusion Policies". In: *Robotics: Science and Systems*. 2023.

[213] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. "Deep Imitative Models for Flexible Inference, Planning, and Control". In: *International Conference on Learning Representations*. 2020.

[214] Robin Rombach et al. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[215] Frieda Rong. *Extrapolating to Unnatural Language Processing with GPT-3's In-context Learning: The Good, the Bad, and the Mysterious*. http://ai.stanford.edu/blog/in-context-learning/. Accessed: 2022-06-04. 2021.

[216] C. Rosen and N. Nilsson. "APPLICATION OF INTELLIGENT AUTOMATA TO RECONNAISSANCE." In: *SRI Technical Report*. 1967.

[217] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

[218] Fereshteh Sadeghi et al. "Sim2Real Viewpoint Invariant Visual Servoing by Recurrent Control". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[219] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. "Meta reinforcement learning with latent variable gaussian processes". In: *arXiv preprint arXiv:1803.07551* (2018).

[220] Chitwan Saharia et al. "Palette: Image-to-image diffusion models". In: *ACM SIGGRAPH 2022 Conference Proceedings*. 2022.

[221] Chitwan Saharia et al. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022.

[222] Chitwan Saharia et al. “Photorealistic text-to-image diffusion models with deep language understanding”. In: *Neural Information Processing Systems (NeurIPS)* (2022).

[223] Yash Satsangi et al. “Maximizing Information Gain in Partially Observable Environments via Prediction Reward”. In: *arXiv preprint arXiv:2005.04912* (2020).

[224] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. “Semi-Parametric Topological Memory for Navigation”. In: *International Conference on Learning Representations*. 2018.

[225] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. “Semi-parametric topological memory for navigation”. In: *arXiv preprint arXiv:1803.00653* (2018).

[226] Nikolay Savinov et al. “Episodic curiosity through reachability”. In: *International Conference on Learning Representations (ICLR* (2019).

[227] Tom Schaul et al. “Universal Value Function Approximators”. In: *International Conference on Machine Learning (ICML)*. 2015.

[228] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017.

[229] Amirreza Shaban et al. “Semantic Terrain Classification for Off-Road Autonomous Driving”. In: *Conference on Robot Learning (CoRL)*. 2022.

[230] Nur Muhammad Mahi Shafiullah et al. “Behavior Transformers: Cloning k modes with one stone”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Ed. by Alice H. Oh et al. 2022.

[231] Nur Muhammad Mahi Shafiullah et al. *CLIP-Fields: Weakly Supervised Semantic Fields for Robotic Memory*. 2023.

[232] Dhruv Shah and Sergey Levine. “ViKiNG: Vision-Based Kilometer-Scale Navigation with Geographic Hints”. In: *Robotics: Science and Systems (RSS)*. 2022.

[233] Dhruv Shah et al. “FastRLAP: A System for Learning High-Speed Driving via Deep RL and Autonomous Practicing”. In: *Annual Conference on Robot Learning (CoRL)*. 2023.

[234] Dhruv Shah et al. “GNM: A General Navigation Model to Drive Any Robot”. In: *International Conference on Robotics and Automation (ICRA)*. 2023.

[235] Dhruv Shah et al. “LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action”. In: *Annual Conference on Robot Learning (CoRL)*. 2022.

[236] Dhruv Shah et al. “Navigation with Large Language Models: Semantic Guesswork as a Heuristic for Planning”. In: *7th Annual Conference on Robot Learning (CoRL)*. 2023.

[237] Dhruv Shah et al. “Offline Reinforcement Learning for Customizable Visual Navigation”. In: *Conference on Robot Learning (CoRL)*. 2022.

[238] Dhruv Shah et al. "Rapid Exploration for Open-World Navigation with Latent Goal Models". In: *5th Annual Conference on Robot Learning*. 2021.

[239] Dhruv Shah et al. "Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action". In: *Conference on Robot Learning (CoRL*. 2022.

[240] Dhruv Shah et al. "Value Function Spaces: Skill-Centric State Abstractions for Long-Horizon Reasoning". In: *International Conference on Learning Representations (ICLR)*. 2022.

[241] Dhruv Shah et al. "ViNG: Learning Open-World Navigation with Visual Goals". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[242] Dhruv Shah et al. "ViNT: A Foundation Model for Visual Navigation". In: *7th Annual Conference on Robot Learning (CoRL)*. 2023.

[243] Nobuyuki Shimizu and Andrew Haas. "Learning to Follow Navigational Route Instructions". In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI'09. 2009.

[244] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. "CLIPort: What and Where Pathways for Robotic Manipulation". In: *Proceedings of the 5th Conference on Robot Learning (CoRL)*. 2021.

[245] Mohit Shridhar et al. "ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[246] R. Sim and J. J. Little. "Autonomous vision-based exploration and mapping using hybrid maps and Rao-Blackwellised particle filters". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2006.

[247] Nathaniel Simon and Anirudha Majumdar. "MonoNav: MAV Navigation via Monocular Depth Estimation and Reconstruction". In: *International Symposium on Experimental Robotics (ISER)*. 2023.

[248] Avi Singh et al. "Cog: Connecting new skills to past experience with offline reinforcement learning". In: *arXiv preprint arXiv:2010.14500* (2020).

[249] Ishika Singh et al. *ProgPrompt: Generating Situated Robot Task Plans using Large Language Models*. 2022.

[250] D. Singh Chaplot et al. "Neural Topological SLAM for Visual Navigation". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[251] Boris Sofman et al. "Improving Robot Navigation Through Self-Supervised Online Learning". In: *Journal of Field Robotics: Special Issue on Machine Learning Based Robotics in Unstructured Environments* (2006).

[252] Haoyu Song et al. "CLIP Models are Few-Shot Learners: Empirical Studies on VQA and Visual Entailment". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2022.

[253] Jiaming Song, Chenlin Meng, and Stefano Ermon. “Denoising diffusion implicit models”. In: *arXiv preprint arXiv:2010.02502* (2020).

[254] Ajay Sridhar et al. “NoMaD: Goal Masked Diffusion Policies for Navigation and Exploration”. In: *arXiv*. 2023.

[255] Rupesh Kumar Srivastava et al. “Training agents using upside-down reinforcement learning”. In: *arXiv preprint arXiv:1912.02877* (2019).

[256] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. “Incentivizing exploration in reinforcement learning with deep predictive models”. In: *arXiv preprint arXiv:1507.00814* (2015).

[257] Hao Sun et al. “Policy continuation with hindsight inverse dynamics”. In: *arXiv preprint arXiv:1910.14055* (2019).

[258] Lauri Suomela et al. “PlaceNav: Topological Navigation through Place Recognition”. In: *arXiv* (2023).

[259] Richard S. Sutton. “Learning to predict by the methods of temporal differences”. In: *Machine Learning* (1988).

[260] Wennie Tabib et al. “Real-Time Information-Theoretic Exploration with Gaussian Mixture Model Maps.” In: *Robotics: Science and Systems*. 2019.

[261] Mingxing Tan and Quoc Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks”. In: *International Conference on Machine Learning (ICML)*. 2019.

[262] Matthew E Taylor and Peter Stone. “Cross-domain transfer for reinforcement learning”. In: *Proceedings of the 24th international conference on Machine learning*. 2007.

[263] Stefanie Tellex et al. “Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation”. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.

[264] Romal Thoppilan et al. “LaMDA: Language Models for Dialog Applications”. In: *CoRR* (2022).

[265] Charles Thorpe et al. “Vision and navigation for the Carnegie-Mellon Navlab”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1988).

[266] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. “Probalistic robotics”. In: *Kybernetes* (2006).

[267] Sebastian Thrun et al. “Stanley: The robot that won the DARPA Grand Challenge”. In: *Journal of field Robotics* (2006).

[268] Naftali Tishby, Fernando C Pereira, and William Bialek. “The information bottleneck method”. In: *arXiv preprint physics/0004057* (2000).

[269] Samuel Triest et al. “TartanDrive: A Large-Scale Dataset for Learning Off-Road Dynamics Models”. In: *International Conference on Robotics and Automation (ICRA)*. 2022.

[270] Joanne Truong et al. “IndoorSim-to-OutdoorReal: Learning to Navigate Outdoors without any Outdoor Experience”. In: *arXiv preprint arXiv:2305.01098* (2023). arXiv: 2305.01098 [cs.RO].

[271] Chris Urmson et al. “Autonomous driving in urban environments: Boss and the urban challenge”. In: *Journal of Field Robotics* (2008).

[272] Karthik Valmeekam et al. *Large Language Models Still Can’t Plan (A Benchmark for LLMs on Planning and Reasoning about Change)*. 2023.

[273] Francisco J. Varela, Evan Thompson, and Eleanor Rosch. *The Embodied Mind: Cognitive Science and Human Experience*. MIT Press, 1991.

[274] Arun Balajee Vasudevan, Dengxin Dai, and Luc Van Gool. “Talk2Nav: Long-Range Vision-and-Language Navigation with Dual Attention and Spatial Memory”. In: *Int. J. Comput. Vision* (2021).

[275] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. 2017.

[276] Ben Wang and Aran Komatsuzaki. *GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model*. https://github.com/kingoflolz/mesh-transformer-jax. 2021.

[277] Yan Wang et al. “Pseudo-LiDAR From Visual Depth Estimation: Bridging the Gap in 3D Object Detection for Autonomous Driving”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[278] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. “Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning”. In: *The Eleventh International Conference on Learning Representations (ICLR)*. 2023.

[279] Saim Wani et al. “MultiON: Benchmarking Semantic Map Memory using Multi-Object Navigation”. In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[280] Jason Wei et al. “Finetuned Language Models are Zero-Shot Learners”. In: *International Conference on Learning Representations (ICLR)*. 2022.

[281] Jason Wei et al. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. In: *Neural Information Processing Systems (NeurIPS)*. 2022.

[282] Jan M. Wiener, Simon J. Büchner, and Christoph Hölscher. “Taxonomy of Human Wayfinding Tasks: A Knowledge-Based Approach”. In: *Spatial Cognition & Computation* (2009).

[283] Erik Wijmans et al. “DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames”. In: *International Conference on Learning Representations (ICLR)*. 2020.

[284] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* (2019).

[285] Yuk Wah Wong and Raymond Mooney. "Learning for Semantic Parsing with Statistical Machine Translation". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. 2006.

[286] Yuxin Wu et al. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[287] Yaqi Xie et al. "Translating natural language to planning goals with large-language models". In: *arXiv preprint arXiv:2302.05128* (2023).

[288] Karmesh Yadav et al. *Habitat Challenge 2022*. https://aihabitat.org/challenge/2022/. 2022.

[289] Karmesh Yadav et al. *Offline Visual Representation Learning for Embodied Navigation*. 2022.

[290] Karmesh Yadav et al. *OVRL-V2: A simple state-of-art baseline for ImageNav and ObjectNav*. 2023. eprint: 2303.07798.

[291] B. Yamauchi. "A frontier-based approach for autonomous exploration". In: *IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. 1997.

[292] An Yan et al. *Cross-Lingual Vision-Language Navigation*. 2019.

[293] Jonathan Yang et al. "Pushing the Limits of Cross-Embodiment Learning for Manipulation and Navigation". In: *Robotics: Science and Systems (RSS)*. 2024.

[294] Naoki Yokoyama, Sehoon Ha, and Dhruv Batra. *Success Weighted by Completion Time: A Dynamics-Aware Evaluation Criteria for Embodied Navigation*. 2021. arXiv: 2103.08022 [cs.RO].

[295] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2014.

[296] Bangguo Yu, Hamidreza Kasaei, and Ming Cao. *L3MVN: Leveraging Large Language Models for Visual Target Navigation*. 2023.

[297] Fisher Yu et al. "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[298] Wenhao Yu et al. "Preparing for the Unknown: Learning a Universal Policy with Online System Identification". In: *Robotics: Science and Systems*. 2017.

[299] Andy Zeng et al. "Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language". In: *arXiv* (2022).

[300] Lunjun Zhang, Ge Yang, and Bradly C Stadie. "World Model as a Graph: Learning Latent Landmarks for Planning". In: *arXiv preprint arXiv:2011.12491* (2020).

[301] Xingyi Zhou et al. "Detecting Twenty-Thousand Classes Using Image-Level Supervision". In: *17th European Conference on Computer Vision (ECCV)*. 2022. DOI: 10.1007/978-3-031-20077-9_21.

[302] Y. Zhu et al. "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

Part IV

APPENDICES

APPENDIX A: OPEN-WORLD EXPLORATION WITH LATENT GOAL MODELS

A.1 DATASET

In this work, we emphasize that data collected from prior experience in unrelated environments can be a rich source of supervision, even if the interactions in the dataset are suboptimal. To demonstrate this, we curate a dataset of over 5000 self-supervised trajectories collected over 9 distinct real-world environments. These trajectories capture the interaction of the robot in diverse environments, including phenomena like collisions with obstacles and walls, getting stuck in the mud or pits, or flipping due to bumpy terrain. The dataset contains measurements from a wide range of sensors including a pair of stereo RGB cameras, thermal camera, 2D LiDAR, GPS and IMU to support offline evaluation using an alternative suite of sensors. While a lot of these sensor measurements can be noisy and unreliable, we believe that learning-based techniques coupled with multimodal sensor fusion can provide a lot of benefits in the real-world. This dataset was collected over a span of 18 months, including parts collected by Kahn et al. [124] and Shah et al. [241] for earlier research projects, and exhibits significant variation in appearance due to seasonal and lighting changes.

This dataset is available for download at sites.google.com/view/recon-robot/dataset, along with helper scripts to load and visualize the trajectories.

A.1.1 *Self-Supervised Data Collection and Labeling*

We design the data collection methodology to enable gathering large amounts of diverse data with minimal human intervention. Due to the high cost of gathering data with real-world robotic systems, we choose to use an off-policy learning algorithm in order to be able to gather data using any control policy and train on all of the gathered data. To ensure that the control policy achieves sufficient coverage of the environment while also ensuring that the action sequences executed by the robot are realistic, we use a time-correlated random walk to gather data. A naïve uniform random control policy is inadequate because the robot will primarily drive straight due to the linear and angular velocity action interface of the robot, which will result in both insufficient exploration and unrealistic test time action sequences.

During data collection using the random control policy, the robot requires a mechanism to detect if it is in collision or stuck, and an automated controller to reset itself in order to continue gathering data. We detect collisions in one of two ways, either using the LIDAR to detect when an obstacle is near or the IMU to detect when the robot is stuck due to an obstacle or uneven terrain. We program an automated backup maneuver that drives the robot out of collision (whenever possible) so it can initiate a new trajectory.

We also use these collision detectors as a weak source of supervision by generating *event labels* for the collected trajectories, giving us a self-supervised relabeling pipeline as proposed in BADGR [124]. We consider three different events: collision, bumpiness, and position. A collision event is detected the LIDAR measures an obstacle to be close or, in off-road environments, when the IMU detects a sudden drop in linear acceleration (jerk) and angular velocity magnitudes. A bumpiness event is calculated as occurring when the angular velocity magnitudes measured by the IMU are above a certain threshold. The position is determined by an onboard state estimator that fuses wheel odometry and the IMU to form a local position estimate. Note that all experiments reported in this paper only use the collision labels; these labels are used to dissect the random walks into smooth trajectories that end in collision.

A.1.2 *Environments*

To learn general navigational affordances across a wide range of environments, we curate over 40 hours of trajectories in 9 diverse open-world environments of varying complexity (see Figure 60).

Figure 61 shows the exploration and navigation performance of RECON and the baselines (see Sec. 3.5 for details) on the individual environments. As the environment complexity increases, most methods are not able to explore the environment efficiently to discover the goal. For videos of our system exploring these environments, please check out the supplemental video submission.

A.2 REPRODUCIBILITY

A.2.1 *Algorithmic Components*

The SubgoalNavigate function rolls out the learned policy for a fixed time horizon H to navigate to the desired subgoal latent z_t^w, by querying the decoder $q_\theta(a_t, d_t | z_t^w, o_\tau)$ in an open loop manner. The endpoint of such a rollout is used to update the visitation counts v in the graph $\mathcal{G}$ using the AssociateToVertex subroutine. To nudge the robot to the frontier, we use a heuristic LeastExploredNeighbor routine that uses the visitation counts of the neighbors to identify unexplored areas in the local neighborhood. At the end of each trajectory, the ExpandGraph subroutine is used to update the edge and node sets $\{\mathcal{E}, \mathcal{V}\}$ of the graph $\mathcal{G}$ to update the non-parametric representation of the environment. Pseudocode for these subroutines are given in Alg. 11.

Algorithm 11 Pseudocode for subroutines referenced in the exploration algorithm shown in Alg. 3

```
function SubgoalNavigate($z_t^w; H$)
    trajectory $\leftarrow ()$
    for $t \in [1, \ldots, H]$ do
        trajectory.append$((o_t, a_t, t))$
        $a_t, d_t^g \sim q_\theta(a_t, d_t \mid z_t^w, o_\tau)$          ▷ []Sample action
        $o_t \leftarrow$ Env.step$(a_t)$          ▷ []Execute action
    end for
    $v_H \leftarrow$ AssociateToVertex$(\mathcal{G}, o_H)$
    $v_H$.count $\leftarrow v_H$.count $+ 1$
    $\mathcal{D}_w \leftarrow ((o_t, o_H, a_t, H - t)$ for $(o_t, a_t, t) \in$ trajectory$)$
    return $\mathcal{D}_w, o_H$
end function
function AssociateToVertex($\mathcal{G} = (\mathcal{V}, \mathcal{E}), o_t$)
    $\mathbf{d} \leftarrow$ sort$((\bar{d}_t^v, v)$ for $v \in \mathcal{V})$          ▷ []Predict distances
    $v, d \leftarrow \mathbf{d}[0]$          ▷ []Associate $o_t$ with nearest vertex
    return $v$
end function
function LeastExploredNeighbor($\mathcal{G} = (\mathcal{V}, \mathcal{E}), o_t, \delta_2$)
    $v \leftarrow$ AssociateToVertex$(\mathcal{G}, o_t)$
    $\mathcal{V}_n \leftarrow \{v' : \mathcal{E}(v, v') < \delta_2, v' \in \mathcal{V}\}$          ▷ []Retrieve neighbors
    $\mathbf{c} \leftarrow$ sort$((v'$.count$, v'$.o$)$ for $v' \in \mathcal{V}_n)$
    $v_c, o_c \leftarrow \mathbf{c}[0]$          ▷ []Retrieve neighbor with smallest count
    return $o_c$
end function
procedure ExpandGraph($\mathcal{G} = (\mathcal{V}, \mathcal{E}), o_t$)
    $v_t \leftarrow$ Node$(\text{count} = 1, o = o_t)$          ▷ []Create node for $o_t$
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(v_t, v_g) : \bar{d}_t^g, g \in \mathcal{V}\}$          ▷ []Add edges
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_t\}$          ▷ []Add vertex
end procedure
```

Hyperparam.	Value	Meaning
δ_1	4	Threshold of identification
δ_2	15	Threshold of neighbors
ϵ	10^{-2}	Exploration threshold on prior
β	1.0	Model complexity
γ	10	Epochs to finetune model
H	5 seconds	Horizon to navigate to subgoal

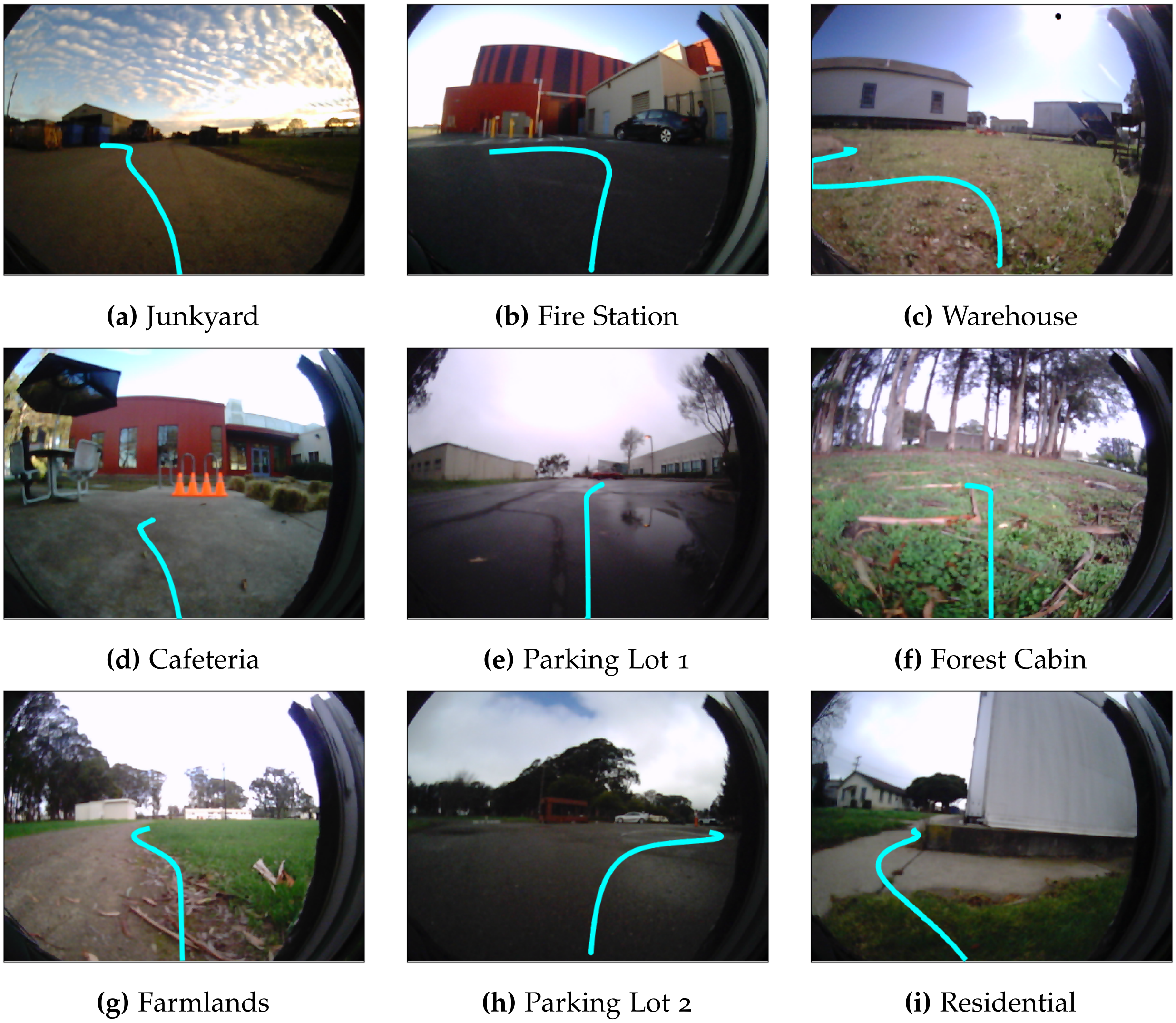
Table 13: Hyperparameters used in our experiments.

A.2.2 *Implementation Details*

Inputs to the encoder p_ϕ are pairs of observations of the environment – current and goal – represented by a stack of two RGB images obtained from the onboard camera at a resolution of 160×120 pixels. p_ϕ is implemented by a MobileNet encoder [104] followed by a fully-connected layer projecting the 1024-dimensional latents to a stochastic, context-conditioned representation z_t^g of the goal that uses 64-dimensions each to represent the mean and diagonal covariance of a Gaussian distribution. Inputs to the decoder q_θ are the context (current observation) – processed with another MobileNet – and z_t^g. We use the reparametrization trick [132] to sample from the latent and use the concatenated encodings to learn the optimal actions a_t^g and distances d_t^g. Details of our network architecture are provided in Table 14. During pretraining, we maximize Eq. 2 with a batch size of 128 and perform gradient updates using the Adam optimizer with learning rate $\lambda = 10^{-4}$ until convergence. We provide the hyperparameters associated with our algorithms in Table 13.

Layer	Input [Dimensions]	Output [Dimensions]	Layer Details
Encoder $p_\phi(z \mid o_t, o_g) = \mathcal{N}(\cdot; \mu_p, \Sigma_p)$			
1	o_t, o_g [3, 160, 120]	I_t^g [6, 160, 120]	Concatenate along channel dimension.
2	I_t^g [6, 160, 120]	E_t^g [1024]	MobileNet Encoder [104]
3	E_t^g [1024]	μ_p [64], σ_p [64]	Fully-Connected Layer, exp activation of σ_p
4	σ_p [64]	Σ_p [64, 64]	torch.diag(σ_p)
Decoder $q_\theta(a, d \mid o_t, z_t^g) = \mathcal{N}(\cdot; \mu_q, \Sigma_q)$			
1	o_t [3, 160, 120]	E_t [1024]	MobileNet Encoder [104]
2	E_t [1024], z_t^g [64]	$F = E_t \oplus z_t^g$ [1088]	Concatenate image and goal representation
3	F [1088]	μ_q [3], σ_q [3]	Fully-Connected Layer, exp activation of σ_q
4	σ_q [3]	Σ_q [3, 3]	torch.diag(σ_q)
5	μ_q [3]	$\bar{a}_t^g[2], \bar{d}_t^g[1]$	Split into actions and distances.

Table 14: Architectural Details of RECON: The inputs to the model are RGB images $o_t \in [0,1]^{3\times160\times120}$ and $o_g \in [0,1]^{3\times160\times120}$, representing the current and goal image.



(a) Junkyard

(b) Fire Station

(c) Warehouse

(d) Cafeteria

(e) Parking Lot 1

(f) Forest Cabin

(g) Farmlands

(h) Parking Lot 2

(i) Residential

Figure 60: We collect data in 9 diverse environments. Example trajectories are shown in cyan.

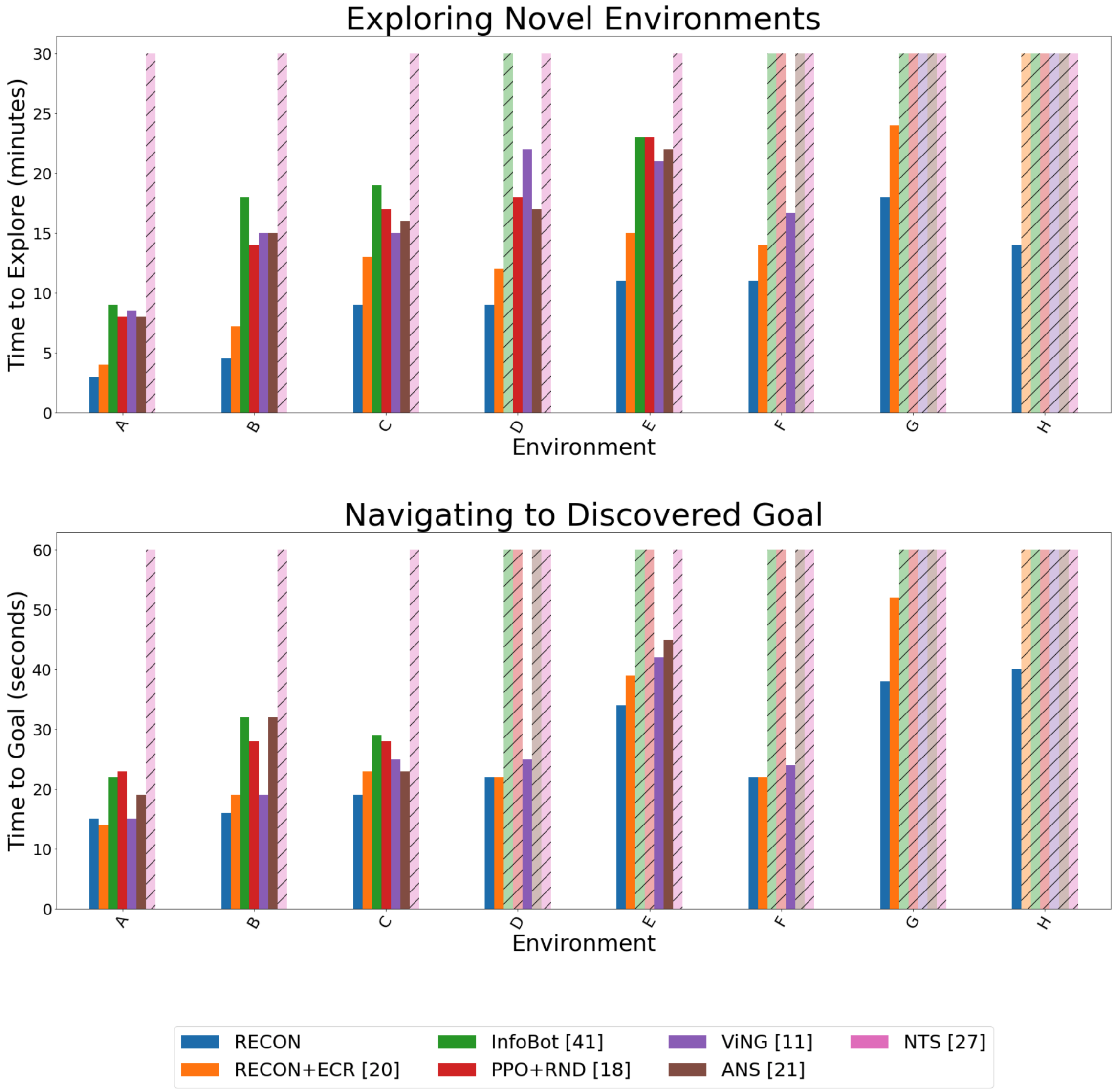


Figure 61: Exploring and learning to reach goals: *(left)* Amount of time needed for each method to search for the goals in a new environment (↓ is better; hashed out bars represent failure). *(right)* Amount of time needed to reach the goal a second time, after reaching the goal once and constructing the map, in seconds (↓ is better).

APPENDIX B: KILOMETER-SCALE EXPLORATION WITH GEOGRAPHIC HINTS

B.1 IMPLEMENTATION DETAILS

Layer	Input [Dimensions]	Output [Dimensions]	Layer Details
		Encoder $p_\phi(z \mid o_t, o_w) = \mathcal{N}(\cdot; \mu_p, \Sigma_p)$	
1	o_t, o_w [3, 160, 120]	I_t^w [6, 160, 120]	Concatenate along channel dimension.
2	I_t^w [6, 160, 120]	E_t^w [1024]	MobileNet Encoder [104]
3	E_t^w [1024]	μ_p [64], σ_p [64]	Fully-Connected Layer, exp activation of σ_p
4	σ_p [64]	Σ_p [64, 64]	torch.diag(σ_p)
		Decoder $q_\theta(a, d, x \mid o_t, z_t^w) = \mathcal{N}(\cdot; \mu_q, \Sigma_q)$	
1	o_t [3, 160, 120]	E_t [1024]	MobileNet Encoder [104]
2	E_t [1024], z_t^w [64]	$F = E_t \oplus z_t^w$ [1088]	Concatenate image and goal representation
3	F [1088]	μ_q [3], σ_q [3]	Fully-Connected Layer, exp activation of σ_q
4	σ_q [5]	Σ_q [5, 5]	torch.diag(σ_q)
5	μ_q [5]	$\bar{a}_t^w[2], \bar{d}_t^w[1], \bar{x}_t^w[2]$	Split into actions, distances and offsets

Table 15: Architectural details of the latent goal model (Section 4.3.1)

B.1.1 *Latent Goal Model (Section 4.3.1)*

Inputs to the encoder p_ϕ are pairs of observations of the environment—current and goal—represented by a stack of two RGB images obtained from the onboard camera at a resolution of 160×120 pixels. p_ϕ is implemented by a MobileNet encoder [104] followed by a fully-connected layer projecting the 1024-dimensional latents to a stochastic, context-conditioned representation z_t^w of the goal that uses 64-dimensions each to represent the mean and diagonal covariance of a Gaussian distribution. Inputs to the decoder q_θ are the context (current observation)—processed with another MobileNet—and z_t^w. We use the reparametrization trick [132] to sample from the latent and use the concatenated encodings to learn the optimal actions a_t^w, temporal distances d_t^w and spatial offsets x_t^w. Details of our network architecture are provided in Table 15. During pretraining, we maximize $\mathcal{L}_{\text{VIB}}$ (Eq. 3) with a batch size of 128 and perform gradient updates using the Adam optimizer with learning rate $\lambda = 10^{-4}$ until convergence.

B.1.2 *Learned Heuristic (Section 4.3.3)*

Inputs to the encoder p_{over} are (i) satellite image c_S and (ii) the triplet of GPS locations $\{x_w, x_S, x_G\}$. p_{over} is implemented as a multi-input neural network with a MobileNet encoder [104] to featurize c_S, which is then concatenated with the location inputs. This is followed by a series of fully-connected layers [512, 128, 32, 1] down to a single cell to predict the binary classification scores. During pretraining, we minimize $\mathcal{L}_{\text{NCE}}$ with a batch size of 256 and perform gradient updates using the Adam optimizer with learning rate $\lambda = 10^{-4}$ until convergence.

B.1.3 *Miscellaneous Hyperparameters*

We provide the hyperparameters associated with our algorithms in Table 16.

Hyperparameter	Value	Meaning
Δt	0.5	Time step of the robot (s)
ϵ	10	Threshold for close (Sec. 4.3.2)
C	20	Scaling constant for v (Alg. 5 L15)
λ_{over}	200	Scaling constant for h_{over} (Sec. 4.3.3)

Table 16: Hyperparameters used in our experiments.

B.2 OFFLINE TRAJECTORY DATASET

For the offline dataset discussed in Section 4.4.2, we use a combination of a 30 hours of autonomously collected data, and 12 hours of human teleoperated data. The complete dataset was collected by 3 independent sets of researchers over the course of 24 months in environments spanning multiple cities. We provide more information below.

B.2.1 *Autonomously Collected Data*

We use the published dataset by Shah et al. [238], that contains over 5000 self-supervised trajectories collected over 9 distinct real-world environments. These trajectories capture the interaction of the robot in diverse environments, including phenomena like collisions with obstacles and walls, getting stuck in the mud or pits, or flipping due to bumpy terrain.

During data collection, a robot is equipped with a 2D LIDAR sensor to detect collisions ahead of time and generate autonomous pseudo-labels for collision events. To ensure that the control policy achieves sufficient coverage of the environment while also ensuring

	Training Dataset	ViKiNG Deployment
Avg. Length	45m	>1km
Avg. Velocity (m/s)	1.68	1.36

Table 17: Trajectory statistics for offline training dataset and real-world deployment.

Environment Type	Amount of Data (hrs)
Paved Hiking Trails	01:45
City Sidewalks	02:15
Suburban Neighborhood Roads	01:30
Unpaved Grasslands	01:00
University/Office Campus	02:30
Miscellaneous	03:00
Total	**12:00**

Table 18: Approximate composition of various environment types in the teleoperated dataset.

that the action sequences executed by the robot are realistic, we use a time-correlated random walk to gather data.

B.2.2 *Human Teleoperated Data*

The above dataset contains extremely diverse dataset that is great for learning general notions of traversability and collision avoidance. However, the random nature of the dataset means that it does not contain any semantically interesting behavior that may be desired of a robotic system, such as following a sidewalk or through a patch of trees. To enhance the quality of learned behaviors, we augment this dataset with about 12 hours of human teleoperated data in semantically rich environments such as hiking trails, city sidewalks, parking lots and suburban neighborhoods. These environments represent realistic scenarios where such a robotic system would be deployed.

Table 17 summarizes key statistics of the trajectories, such as length and velocity. Table 18 summarizes the various environments in which the dataset was collected, and their relative composition. Figure 62 visualizes the geographic locations of these data collection sites (location anonymized for the double-blind review process). We ensure no overlap between the training and test environments—success in these test environments requires *true* generalization to unseen environments.

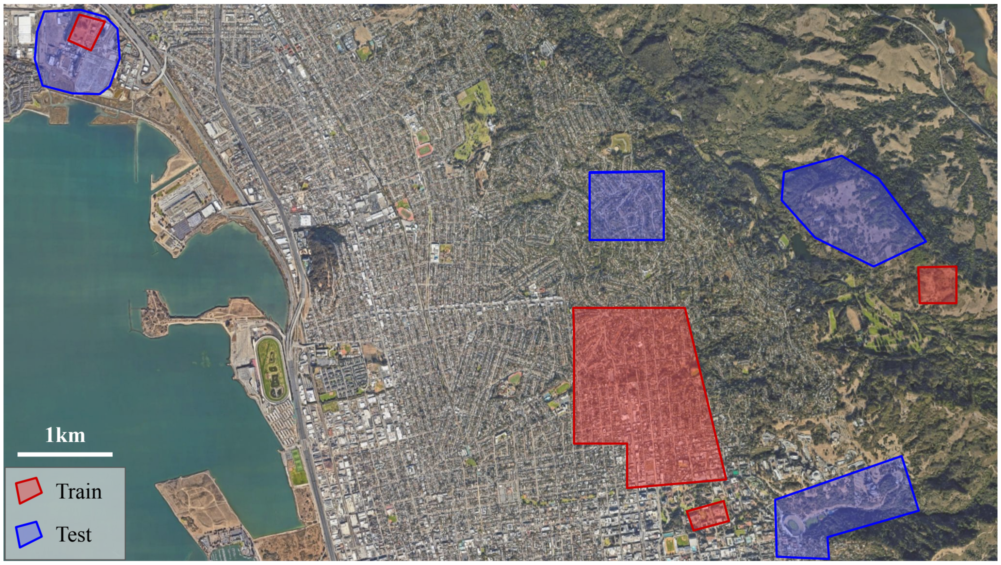


Figure 62: Rough geographical locations of data collection by human teleoperation and testing (Section 4.4)

APPENDIX C: OFFLINE REINFORCEMENT LEARNING FOR VISUAL NAVIGATION

C.1 FORMAL ANALYSIS OF PROPOSITION 3.1

Proposition 3.1 *If we recover the optimal value function $V^*(s, s')$ for short-horizon goals s' (relative to s), and $\mathcal{G} = \mathcal{S}$ (all states exist in the graph), and the MDP is deterministic with $\gamma = 1$, then finding the minimum-cost path in the graph $\mathcal{G}$ with edge-weights $-V^*(s, s')$ recovers the optimal path. .*

Proof: Let $A(s)$ and $A^h(s)$ define a set of all nodes adjacent to node s and within a short horizon from a node s correspondingly.

The Bellman equation can be used to write the cost of the minimal-cost path, $J^*(s, g)$, in the graph with rewards defined via edge-weights $r(s, a, s') = -V^*(s, s')$:

$$J^*(s,g) = \min_{s' \in A^h(s)} [-V^*(s,s') + J^*(s',g)] = -\max_{s' \in A^h(s)} [V^*(s,s') - J^*(s',g)].$$

We can expand the recursion:

$$J^*(s,g) = -\max_{s' \in A^h(s), s'' \in A^h(s'), \ldots, g \in A^h(s^{(n)})} [V^*(s,s') + V^*(s',s'') + \ldots + V^*(s^{(n)},g)]. \quad (16)$$

We can further expand each $V^*(\cdot, \cdot)$ term as

$$V^*(s^{(n-k)}, s^{(n-k+1)}) = \max_{\substack{s_1 \in A(s^{(n-k)}) \\ s_2 \in A(s_1) \\ \cdots \\ s^{(n-k+1)} \in A(s_t) \\ t \in \mathbb{N}}} [-C(s^{(n-k)}, s_1) - C(s_1, s_2) - \ldots - C(s_t, s^{(n-k+1)})]. \quad (17)$$

If we expand every term in Equation 16 with 17 it becomes exactly the optimization objective for the shortest path problem with the original edge-weights. One can see $V^*(s, s')$ as a solution to the shortest path problems in the subgraphs of $\mathcal{G}$ induced by $A^h(s)$.

C.2 REWARD LABELING

For the base task of goal-reaching, we use a simple reward scheme with a *survival penalty* that incentivizes the robot to take the shortest path to the goal:

$$R_{\text{dist}}(s_t, a_t, g) = \begin{cases} -1 & \forall s_t \neq g \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

For more complex utilities, such as incentivizing driving in the sun (e.g., for a solar robot), we discount the survival penalty by a factor of 4.

$$R_{\text{grass}}(s_t, a_t, g) = \begin{cases} -1 + 0.75 * \mathbb{1}_{\text{grass}}\{s_t\} & \forall s_t \neq g \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

$$R_{\text{sun}}(s_t, a_t, g) = \begin{cases} -1 + 0.75 * \mathbb{1}_{\text{sun}}\{s_t\} & \forall s_t \neq g \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

An interesting implication of the above reward scheme is to view the negative penalty as a proxy for the amount of work a robot needs to do — a solar robot may use 1 unit of energy per time step to navigate in an environment, but it may also create 0.75 units of energy by exposing itself to the sun, effectively discounting the navigation cost in sunny regions. This reward scheme trades-off the choice of the shortest path to the goal with maximizing the user-specified utility function.

For our experiments, we use three different mechanisms to generate these labels:

1. **Fully Autonomous:** In several cases, the reward signal can be easily expressed as a linear/heuristic function of the visual observations. For instance, to obtain labels for "sunny" or "grassy", we process the egocentric images from the robot by thresholding in the HSV colorspace. We process the bottom center crop of the image by thresholding it, and declare event $\mathbb{1}_{\text{sun}}$ or $\mathbb{1}_{\text{sun}}$ if a majority of the pixels satisfy the thresholds.

2. **Manual Labeling:** For more abstract tasks, generating reward labels may require careful hand-labeling at the level of each observation, or each frame. We generate labels for "on-sidewalk" by manually labeling trajectories that are driving on the sidewalk/pavement — this was only feasible because the number of such trajectories was relatively small.

3. **Learned Reward Classifiers:** A desirable hybrid of the above approaches can be constructed where manual labels are queried for a small portion of the training dataset, which can be used to train a simple image classification model. This model can be used to obtain reward labels, albeit noisy, for the remainder of the dataset in a *semi-autonomous* way. We follow this process for obtaining high-quality labels for the "grassy" and "on-pavement" tasks. So long as the reward signal is fully contained by the visual observation, which loosely relates to the Markovian assumption for RL, this method gives us a scalable way to learn a predictive model of rewards.

We note that while the above choice of reward function may seem arbitrary, the overall utility function (or the "relative weight" between the two objectives) would be

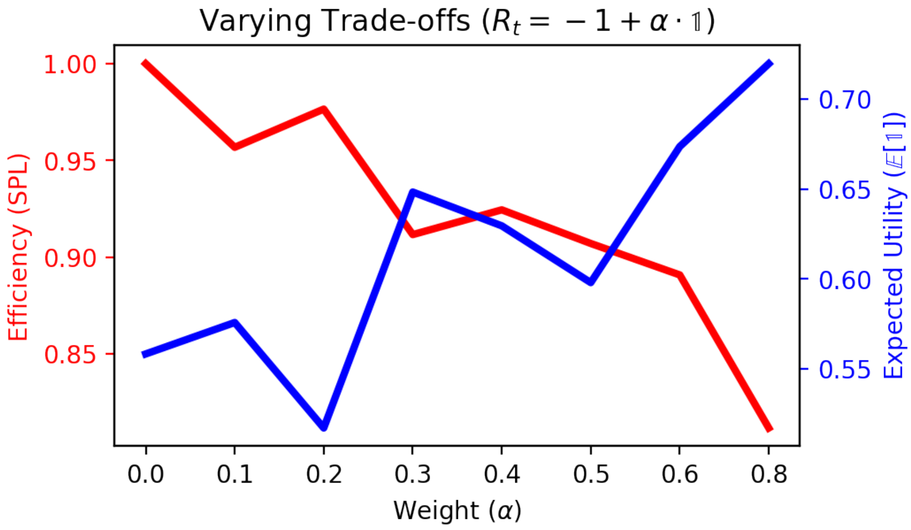


Figure 63: ReViND can support a wide range of reward functions and performs as expected for varying levels of trade-offs between the goal-reaching and utility maximization objectives.

application-dependent. For instance, a solar-powered robot may be able to recoup 20% of its navigation energy when driving in the sun, and its effective reward could be $(-1 + 0.2 * \mathbb{1}_{\text{sun}})$. We ran experiments to test ReViND's sensitivity to this trade-off and found that it performs expectedly for a wide range of reward functions (see Figure 63). Practically, this would be a hyperparameter set empirically by the user based on the desired level of trade-off between the goal-reaching and utility maximization objectives.

C.3 BUILDING THE TOPOLOGICAL GRAPH

As discussed in Section 5.3.3, we combine the value function learned via offline RL with a topological graph of the environment. This section outlines the finer details regarding how this graph is constructed. We use a combination of learned value function (from Q-learning), spatial proximity (from GPS), and temporal proximity (during data collection), to deduce edge connectivity. If the corresponding timestamps of two nodes are close ($< 2s$), suggesting that they were captured in quick succession, then the corresponding nodes are connected — adding edges that were physically traversed. If the distance estimates (or, negative value) between two nodes are small, suggesting that they are *close*, then the corresponding nodes are also connected — adding edges between distant nodes along the same route, and giving us a mechanism to connect nodes that were collected in different trajectories or at different times of day but correspond to the nearby locations. To avoid cases of underestimated distances by the model due to aliased observations, e.g. green open fields or a white wall, we filter out prospective edges that are significantly further away as per their GPS estimates — thus, if two nodes are nearby as per their GPS, e.g. nodes on different sides of a wall, they may not be disconnected if the values do not estimate a small distance; but two similar looking nodes 100s of meters away, that may be facing a white wall, may have a small distance estimate but are not added to the graph in order to avoid *wormholes*. Algorithm 0 summarizes this process — the timestamp threshold ϵ is 1 second, the learned distance threshold τ is 50 time steps (corresponding to ~ 12 meters), and the spatial threshold η is 100 meters.

Algorithm 12 Graph Building

```
function GetEdge($(i, j)$)
    Input: Nodes $n_i, n_j \in \mathcal{G}$; value function $V_\psi$; hyperparameters $\{\tau, \epsilon, \eta\}$
    Output: Boolean $e_{ij}$ corresponding to the existence of edge in $\mathcal{G}$, and its weight
    goal = GetRelative$(n_i, n_j)$                              ▷ using GPS and compass
    $D_{ij} = -V_\psi(n_i, \text{goal})$                          ▷ learned distance estimate
    $T_{ij} = |n_i[\text{'timestamp'}] - n_j[\text{'timestamp'}]|$    ▷ timestamp distance
    $X_{ij} = \|n_i[\text{'GPS'}] - n_j[\text{'GPS'}])\|$             ▷ spatial distance
    if ( $T_{ij} < \epsilon$) then return {True, $D_{ij}$}
    else if ($D_{ij} < \tau$) AND ($X_{ij} < \eta$) then return {True, $D_{ij}$}
    else return False
end function
```

Since a graph obtained by such an analysis may be quite dense, we perform a *transitive reduction* operation on the graph to remove redundant edges.

C.4 EXTENDED EXPERIMENTS/BASELINES

This section presents a detailed breakdown of the quantitative results discussed in Section 5.4.3. We evaluate ReViND against four baselines in 15 environments with varying levels of complexity, in terms of environment organization, obstacles, and scale. Tables 8 and 19 summarize the performance of the different methods for the task of maximizing the R_{grass} and R_{sun} utilities, respectively.

We see that ReViND is able to consistently outperform the baselines, both in terms of success as well as its ability to maximize the utilities $\mathbb{1}_{.}$. In particular, we see that ReViND's performance closely matches that of IQL in the easier environments, where the system does not need to rely excessively on the graph. However, the real prominence of ReViND is evident in the more challenging environments, where it is consistency successful while also maximizing the chosen utility. As the horizon of the task increases, the search algorithm on the graph returns more desirable paths that may stray from the direct, shortest path to the goal, but are highly effective in maximizing the utility. We also note that ViNG, which uses a similar topological graph, is statistically similar to ReViND in terms of its goal-reaching ability; however, since it does not support a mechanism to

Method	Easy (<50m)		Medium (50–150m)		Hard (150–500m)	
	Success	$\mathbb{E}\mathbb{1}_{\text{sun}}$	Success	$\mathbb{E}\mathbb{1}_{\text{sun}}$	Success	$\mathbb{E}\mathbb{1}_{\text{sun}}$
Behavior Cloning	1/5	0.58	0/5	0.32	0/5	0.29
Filtered BC	3/5	0.51	0/5	0.31	0/5	0.32
IQL [137]	3/5	0.54	2/5	0.42	0/5	0.34
ViNG [241]	**5/5**	**0.63**	**4/5**	0.58	3/5	0.63
ReViND (Ours)	**5/5**	0.61	3/5	**0.75**	**4/5**	**0.74**

Table 19: Success rates and utility maximization for the task of navigation in sunny regions (R_{sun}).

customize the behavior of the learned policy, it suffers in the other performance metrics. BC, fBC and IQL consistently fail to reach goals beyond 15-20m away, due to challenges in learning a useful policy from offline data — these "flat" policies often demonstrate *bee-lining* behavior, driving straight to the goal, which leads to collisions in all but the easiest experiments.

C.5 MISCELLANEOUS IMPLEMENTATION DETAILS

Table 20 presents the neural network architectures used by our system. We provide the important hyperparameters for training our system in Table 21. The underlying learning algorithm in ReViND is based on IQL [137], and we encourage the reader to check out the IQL paper for more implementation details.

C.6 ENVIRONMENTS

We train ReViND using 30 hours of publicly available robot trajectories collected using a randomized data collection procedure in an office park [238]. We conduct evaluation experiments in a variety of novel environments with similar visual structure and composition as the training environments — i.e. suburban environments with some traversals on the grass, around trees of a certain kind, and on roads. While it may be extremely challenging to get generalization capabilities that work for *all* scenarios, we demonstrate that ReViND can learn behaviors from a small offline dataset and generalize to a variety of previously unseen, *visually similar* environments including grasslands, forests and suburban neighborhoods. Figure 64 shows some example environments from the training and deployment environments, along with their corresponding labels (automatically generated).

Layer	Input Shape	Output Shape	Layer details
1	[3, 64, 48]	[1536]	Impala Encoder [64]
2	[1536]	[50]	Dense Layer
3	[50]	[50]	$\tanh$ (LayerNorm)
4	[50], [3]	[53]	Concat. image & goal
Policy Network $a_t \sim \pi(s_t)$			
5	[53]	[256]	Dense Layer
6	[256]	[256]	Dense Layer
7	[256]	[10]	Dense Layer
Q Network $Q(s_t, a_t)$			
5	[53], [10]	[256]	Dense Layer
6	[256]	[256]	Dense Layer
7	[256]	[1]	Dense Layer
Value Network $V(s_t)$			
5	[53]	[256]	Dense Layer
6	[256]	[256]	Dense Layer
7	[256]	[1]	Dense Layer

Table 20: Architectures of the various neural networks used by ReViND.

Hyperparameter	Value	Meaning
τ	0.9	IQL Expectile
A	0.1	Policy weight
γ	0.99	Discount factor
η	0.005	Soft Target Update
$\alpha_{\text{actor}}, \alpha_{\text{critic}}, \alpha_{\text{value}}$	$3e-4$	Learning rates

Table 21: Hyperparameters used during training ReViND from offline data.

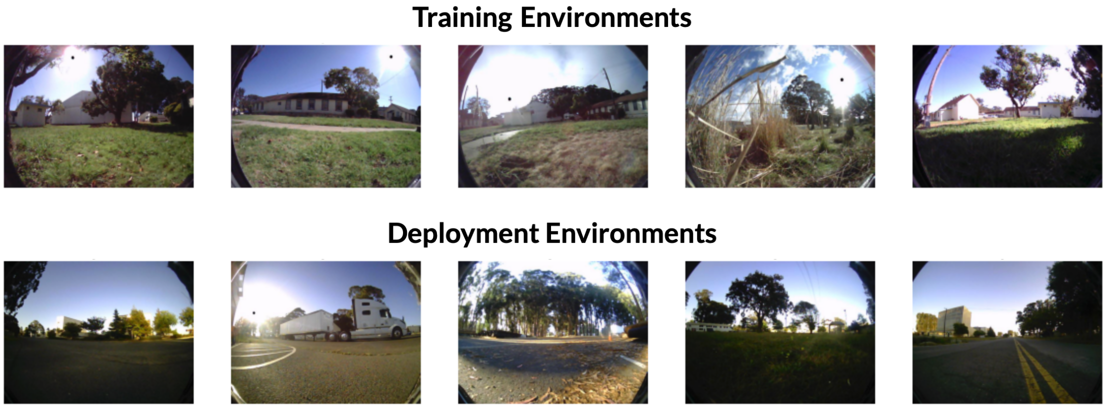


Figure 64: Example egocentric observations from the training dataset [238] (top) and the deployment environments (bottom), including the predicted labels for the "sunny" reward.

APPENDIX D: A FOUNDATION MODEL FOR VISUAL NAVIGATION

D.1 VINT MODEL ARCHITECTURE

Table 22 shows the ViNT model architecture in detail. We use all 18 layers of the EfficientNet-B0 convolutional encoder [261], initialized from scratch, to tokenize the observation and subgoal images into 512-dimensional embeddings each. We utilize an observation encoder to tokenize the past and current observation images and a joint observation and goal encoder to tokenize the subgoal image fused with the current observation image channel-wise. For tokenizing the joint observation and subgoal token, images o_t and o_s are concatenated along their channel dimension, yielding a $6 \times 85 \times 64$ tensor per training data point.

Layer	Input [Dimensions]	Output [Dimensions]	Layer Details
1	o_t, o_s [64, 85, 3]	I_t^g [64, 85, 6]	Concatenate observations and goal
2	I_t^s [64, 85, 6]	E_t^s [1, 1000]	Goal EfficientNet encoder
3	$o_{t:t-P}$ [P+1, 64, 85, 3]	$E_{t:t-P}$ [P+1, 1000]	Context EfficientNet encoder
4	E_t^s [1, 1000]	$E_t^{s'}$ [1, 512]	Goal embedding compression
5	$E_{t:t-P}$ [P+1, 1000]	$E'_{t:t-P}$ [P+1, 512]	Context embedding compression
6	$E'_{t:t-P}$ [P+1, 512], $E_t^{s'}$ [1, 512]	S [P+2, 512]	Concatenate
7	S [P+2, 512]	$\tilde{S}$ [1, 32]	Feed into Transformer f
8	$\tilde{S}$ [1, 32]	d	Predict temporal distance d
9	$\tilde{S}$ [1, 32]	$\hat{a}$, [1, T, 4]	Predict future actions $\hat{a}$

Table 22: Architectural Details of ViNT The inputs to the model are RGB images $o_{t:t-P} \in [0,1]^{P \times 3 \times 85 \times 64}$ and $o_s \in [0,1]^{3 \times 85 \times 64}$, representing the current, past, and goal images. We seek to predict a H future actions $\hat{a}$ and the temporal distance d.

D.1.1 *Goal-Conditioning Architectures*

We considered different mechanisms for conditioning ViNT with features from the subgoal image, as illustrated in Figure 65.

1. **Late Fusion:** Extract the observation and goal features independently and fuse them in the multi-head attention layers. To achieve this effect, we avoid any channel-wise concatenation between any of the observation and goal images before inputting them into the model.

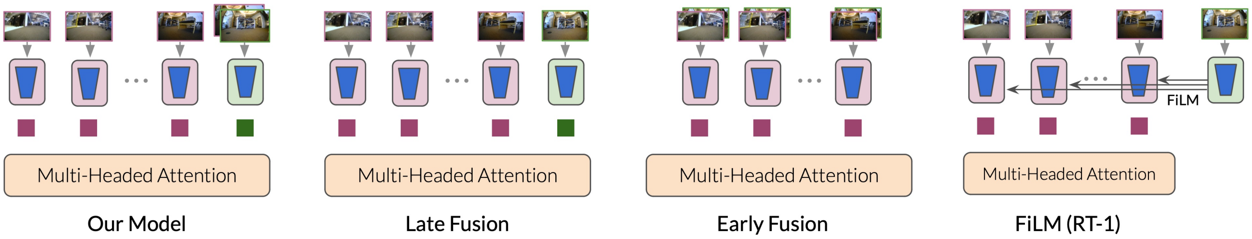


Figure 65: Different goal-conditioning architectures considered for ViNT.

2. **Early Fusion:** Jointly extract observation (context) and goal features and fuse the observation and goal features before we tokenize them. We achieve this by concatenating the goal image with every observation image along the channel dimension. We remove the goal token in this setup since information about the goal is already in every observation token.

3. **FiLM (RT-1):** Following the FiLM+EfficientNet encoder ([21]), encode each observation image separately. For conditioning on visual goals, we replace the "Universal Sentence Encoder" with an EfficientNet encoder. We remove the goal token in this setup since information about the goal is already in every observation token.

Our observations are summarized in Table 23. While FiLM works well for language, we found that training was unstable for image-based navigation tasks. Instead, we directly encode each observation independently and pass them to a Transformer. Ideally, the goal would be encoded separately and then combined with the observations in the Transformer layers, allowing the entire goal encoder to later be swapped out for different goal modalities. Unfortunately, we found that this approach (which we term "late fusion", as the goal and observations are not fused until *after* encoding them) performs poorly: in image-based navigation, it is the *relative* features between the observation and goal images that are important, rather than *absolute* goal features. An "early fusion" architecture would fuse the goal image with all the past and current observation images immediately, which allows for learning joint features between the goal image and current state. However, this architecture is inflexible as the observation encoder would have to be learned entirely from scratch when adapting to a new goal modality. ViNT avoids this issue by using two distinct types of encoders: an observation-only encoder used to tokenize each observation image, and a joint observation and goal encoder that should extract relative goal features. This latter encoder can be replaced to allow alternative goal specifications in downstream tasks, as described in Appendix D.2.4. Specifically, we adapt to new tasks by learning the final token conditioned on the new task goal information in place of the joint observation/goal encoder.

Method	Performance	Adaptation
Late Fusion	✗	✓
Early Fusion	✓	✗
FiLM (RT-1) [21]	✗	✓
ViNT	✓	✓

Table 23: Comparing merits (✓) and demerits (✗) of different goal-conditioning architectures. While "Early Fusion" works the best for the core navigation task, it does not support downstream adaptation (Section 7.5). "Late Fusion" is ideal for adaptation, but does not perform well for our tasks. Our goal fusion architecture is able to closely match the performance of early fusion, while also supporting adaptation.

D.2 IMPLEMENTATION DETAILS

D.2.1 *Training ViNT*

See Table 24 for a detailed list of hyperparameters for training the ViNT foundation model.[3]

D.2.2 *Subgoal Diffusion*

For generating subgoals, we use an image-to-image diffusion model. It takes an image o_t as input and produces samples from $g(o_{s_i} \mid o_t)$, where o_{s_i} are candidate subgoal images reachable from o_t. To produce training pairs for the diffusion model, we first select o_t uniformly at random from the training data and then select o_{s_i} to fall between 5 and 20 timesteps in the future from o_t.

Following [220], we implement image conditioning as simple channel-wise concatenation to the U-Net input. We use the Flax U-Net implementation from the diffusers library [202] with textual cross-attention removed since we do not condition on text inputs.

We use the continuous-time diffusion formulation from [131] with a fixed linear noise schedule rather than a learned one. Also unlike [131], we use the unweighted training objective, called L_{simple} in [99, Equation 14] and [131, Appendix K]. We employ classifier-free guidance [100] and find that it helps produce subgoals with better visual fidelity, which is consistent with prior work [222].

[3] We used a variety of workstations equipped with different GPU configurations over the course of this research, including 2×4090, 3×Titan Xp, 4×P100, 8×1080Ti, 8×V100, and 8×A100. With the model architecture fixed, the batch size and training time varies significantly across these devices, and the entry in Table 24 is representative of our most common training configuration.

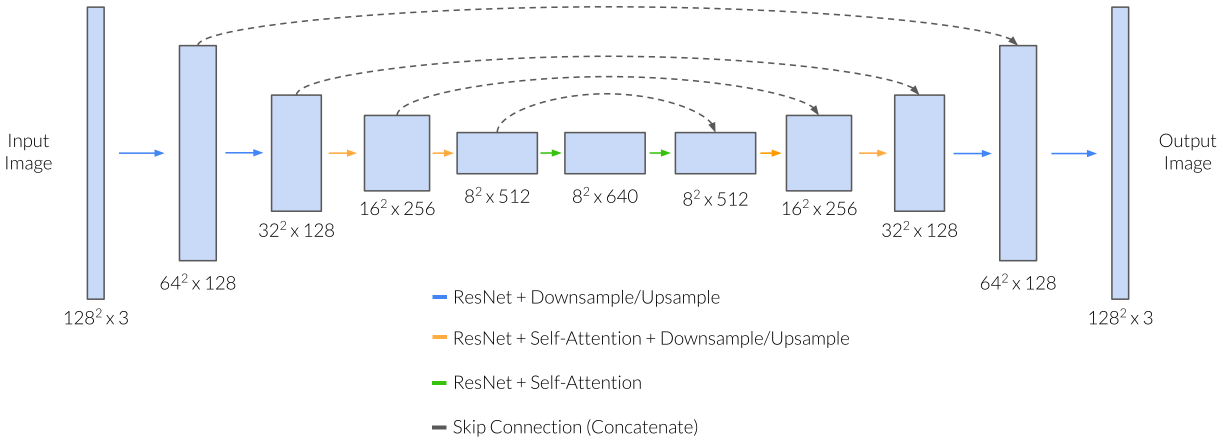


Figure 66: Subgoal diffusion model U-Net architecture. Each ResNet consists of 2 residual blocks. Downsampling and upsampling is done with strided convolutions.

D.2.3 *Long-Horizon Physical Search via Topological Graphs*

As in [232], we implement physical search similarly to a standard A* algorithm, by keeping track of an open set Ω of possible unvisited subgoals (generated by our diffusion model) and following Alg. 13.

Algorithm 13 Long-Horizon Navigation via Topological Graph

```
while goal $G$ not reached do
    $s \leftarrow \min_f(\Omega)$
    $P \leftarrow$ ShortestPath($\mathcal{M}$, $o_t$, $s^-$)
    for $(s, s')$ in $P$ do
        ViNT.GoToGoal($s'$)
    end for
    ViNT.GoToGoal($s$)
    $o_t \leftarrow$ Observe()
    AddNode($\mathcal{M}$, $o_t$, parent: $s^-$)
    Sample $s_i \sim g(s_i|o_t)$
    Add($\Omega$, $s_i$)
end while
```

Nodes are visited according to a costing function $f(s)$ that depends on the distance from the current state o_t to the parent node s^- (measured along the graph), the predicted distance from s^- to s, and a heuristic function h (similar to that of A*) providing long-horizon navigation hints:

$$f(s) = d_{\mathcal{M}}(o_t, s^-) + d_{\text{pred}}(s^-, s) + h(s, G, C)$$

In general, the heuristic can be any function providing a notion of distance between a subgoal s and the long-horizon goal G, optionally with some context C. For our experiments, we considered three heuristics to demonstrate the flexibility of our approach:

- **Coverage exploration:** We have no long-horizon guidance for coverage exploration, and thus, use $h(s) = 0$.
- **Position-guided:** For long-horizon GPS goals (outdoors) and 2D position goals (indoors), we use Euclidean distance $h(s) = \|s - G\|$.
- **Satellite-guided:** In the context-guided experiments, we train a learned heuristic function that uses the satellite image as an input to learn a a heuristic for "good" subgoals. We train a convolutional neural network on the overhead image to predict the probability that the subgoal s is included on a trajectory from o_t to G, trained using a contrastive objective [194]. Additional information can be found in [232].

D.2.4 *Fine-tuning ViNT*

In all CARLA fine-tuning experiments, on-task data was collected using a rule-based oracle agent, with start and end locations sampled randomly up to 900 meters apart. We collect 181 training trajectories (roughly 4 hours) in CARLA's Town 01 environment, and a further 52 trajectories (1 hour) in the held-out Town 02 environment. Inspired by [45], we further augment this dataset by allowing the rule-based agent to correct its position and re-center to the lane after a perturbation.

IMAGE FINE-TUNING:

- **Architecture:** We utilize the exact same architecture as ViNT with no changes.
- **Training:** For fine-tuning the image-goal directed model, we utilize the same training process for ViNT with a learning rate of 0.0001, AdamW optimizer, but no warmup or cosine scheduler. We do not mix any prior data for fine-tuned training.

GPS-ADAPTATION:

- **Architecture:** To adapt to GPS-style goals, we cut off the goal encoder block from ViNT. We then learn a fixed tensor of size 3000 and concatenate it to the GPS-command goal in ego-centric coordinates. We then pass this into a 2-layer MLP which outputs the prediction of the final token for the transformer. The architecture is shown in Figure 67.
- **Training:** During training, instead of randomly sampling future images to serve as goals, we sample goals from future odometry information. Once we have a future goal coordinate for self-supervision, we convert to local coordinates and pass

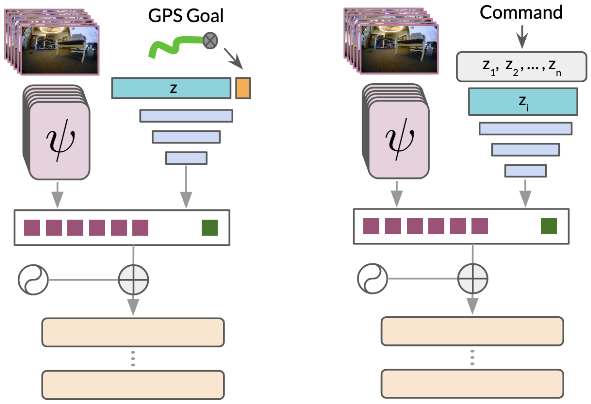


Figure 67: Adaptation architectures for ViNT. Left: GPS-adaptation architecture. The local coordinates of the goal are concatenated to the fixed latent z. Right: command-adaptation architecture, using latent z_i selected by command label index i.

into our architecture, fine-tuning with the same objective as ViNT. We use a cosine scheduler with a learning rate warmup to 0.0001 for 4 epochs. We also sample goal points from between 1.25s and 1.75s rather than from 0.5s to 2.5s.

COMMAND-ADAPTATION:

- **Architecture:** For discrete command goals, we adopt a similar approach for GPS-style goals. We learn a fixed tensor for each discrete command and use the command index to select the corresponding latent to pass into a 2-layer MLP for predicting the final token. In this way, we learn a dictionary of latents, each corresponding to a distinct command. This architecture is illustrated in Figure 67.
- **Training:** For our experiments, we use “left”, “right”, and “straight” as our discrete commands. We assume training data is not labelled with the discrete command, so we label dataset trajectories with the corresponding commands retroactively by sampling a future position (as in GPS-Adaptation) and then selecting a command based on its lateral deviation. For our experiments we bin samples with lateral coordinate greater than 0.05 as “left” or “right” and label the remaining samples as ”straight”. We again use a cosine scheduler with a learning rate warmup to 0.0001 for 4 epochs.

D.3 TRAINING DATASET

The ViNT training dataset contains over 100 hours of real-world navigation trajectories, sourced entirely from existing datasets. The dataset consists of a combination of tele-operated and autonomous navigation behaviors collected across 8 distinct robotic platforms, including 4 commercially available platforms (TurtleBot, Clearpath Jackal, Warthog and Spot) and several custom platforms (Yamaha Viking ATV, RC Car, passenger automobiles). The trajectories contain widely varying robot dynamics and top speeds, ranging between 0.2 and 10m/s, operating in a diverse set of environments (e.g., office buildings, hallways, suburban, off-road trails, university campuses, etc.). All data is either publicly available, or collected by *other* researchers for past projects; *no additional training data was collected specifically for training ViNT*.

Remember to mention: total size, number of robots, conversion to number of frames and so on.

D.4 ROBOTIC PLATFORMS FOR EVALUATING VINT

VIZBOT: A custom-built robot platform inspired by the design of [192], based on a Roomba. It is equipped with an off-the-shelf PCB-mounted fisheye camera.

UNITREE GO 1: A commercially available quadruped robot equipped with the original forward-facing camera. *There is no training data from a Go 1 in the training dataset*. Athough SCAND includes data collected on a Boston Dynamics Spot, which is also a quadrupedal robot, the two platforms practically have very different characteristics.

CLEARPATH JACKAL UGV: A commercially available off-road platform equipped with an off-the-shelf PCB-mounted fisheye camera. *This system resembles the data collection platform used for the RECON, Berkeley, and SCAND-J datasets*, but has a different camera and mounting height.

LOCOBOT: A popular open-source platform based on a Kobuki equipped with an off-the-shelf PCB-mounted fisheye camera. *This robot is not present in the training dataset*, although GS was collected on a similar TurtleBot2 with a different spherical camera at a lower height.

D.5 EVALUATION SETUP AND DETAILS

D.5.1 *Navigation Performance*

Indoor Experiments

For setting up the indoor coverage exploration experiments, we use the LoCoBot and Vizbot robotic platforms. We choose a random starting point and goal in an enclosed environment, and keep these locations consistent across all baselines we test. For the coverage exploration task, we ensure that the environments are "enclosed" and block any glass walls and stairwells, which are beyond the capabilities of the robots. Experiments are terminated when either (i) the robot is unable to reach the goal within a pre-specified time limit of 10 minutes, or (ii) the robot becomes physically stuck (e.g., collides and is unable to recover).

For setting up the indoor guidance exploration experiments on the LoCoBot, we mark the start and goal locations in a large office building and note their 2D positions. The goal location is conveyed to the robot as the context, and is available to the search algorithm. The system uses the robot's onboard wheel odometry to track position.

Outdoor Experiments

For the coverage exploration experiments, we follow the setup of [238] and use the Clearpath Jackal UGV. We choose a random start and goal location in confined outdoor environments and obtain a goal image observation for the robot to seek. Experiments are terminated either when (i) the robot is unable to reach the goal within a pre-specified time limit of 20 minutes, or (ii) the robot collides with an obstacle in the environment.

For the guided exploration experiments, we closely follow the setup of [232]. For the GPS guided experiments, the robot has access to the GPS location of the goal, in addition to a goal image. For the satellite-guided experments, the robot further has access to an overhead satellite image centered at its current location and a learned heuristic funtion h.

Baselines

For experiments presented in Section 7.6.1, we evaluate 4 baselines against our method.

1. **End-to-End BC:** A modified ViNT model with no goal token, trained end-to-end for the task of only predicting future actions. This represents a typical undirected BC baseline with similar model capacity as the other baselines.

2. **End-to-End GCG:** A model-based algorithm that uses a predictive model to plan a sequence of actions that reach the goal without causing collisions [123]. Since this model requires collision labels for training, it is only trained on a subset of the training data (RECON, CoryHall, Berkeley) that has these labels; hence, this baseline is only evaluated outdoors.

3. **RECON:** A variant of the physical search algorithm RECON [238], which uses a latent goal model to represent reachable goals and plans over sampled subgoals to explore the environment in a similar manner to ours. This baseline uses a variational information bottleneck to sample latent subgoals, rather than a diffusion model sampling subgoal images.

4. **ViNT-R:** An ablation of our method that uses subgoals randomly sampled from the training data, instead of samples from a conditional diffusion model, as subgoal candidates.

D.5.2 *Multi-robot Generalization Experiments*

The setup for the multi-robot generalization experiment is same as the coverage exploration experiments. The only differences are the baselines we evaluate.

Baselines

For experiments presented in Section 7.6.2, we test three baseline low-level policies on each robot. Each baseline uses the graph-based exploration scheme described in Section 7.4.1. We use the following baselines:

1. **Single-Robot:** We train a single-dataset policy model (ViNT architecture) and diffusion model on the two largest datasets (RECON for outdoor, and SACSoN for indoor), and evaluate them on each of our robots to identify the best single-dataset model for each robot. Note that we do not have comparable magnitudes of training data of visual locomotion on the Go 1.

2. **GNM:** We use the pre-trained model checkpoint from the authors of GNM [234] coupled with *our* diffusion model (since GNM is not compatible with the exploration task) to evaluate each robot.

3. **ViNT:** We use our pre-trained ViNT policy and image diffusion model (no fine-tuning) to evaluate each robot.

D.5.3 *Fine-tuning and Adaptation*

This section describes the setup and implementation details for ViNT fine-tuning and adaptation experiments in the CARLA autonomous driving simulator, as presented in Sections 7.6.3 and 7.6.4.

CARLA Data Collection

We collect expert trajectories with an oracle rule-based self-driving agent and gather odometry and RGB information across the trajectory at 4 Hz. These trajectories have random spawn points and random destination points up to to 900 meters in length. We collect 52 trajectories in the CARLA Town 02 for held-out testing, and collect 181 trajectories in Town 01 for training. This makes for a dataset size of 5 hours for the autopilot control data. Inspired by [45], we also collect short trajectories of the agent correcting back onto the right lane after drifting off course in Town 01 and Town 02 for training and testing, respectively. This data is 4 hours long, and we add it to the autopilot data for training.

Fine-tuning Experiments

To test the fine-tuning system which trains ViNT with the same goal specification but in new domains, we utilize the collected test trajectories as sequences of goal images to follow. Each Town 02 test trajectory creates a graph in which every node is a timestamped odometry point corresponding to an image. To evaluate a model on a test trajectory, we spawn it at the same start point and localize it on the trajectory's map. We then query the image for the goal node which corresponds to node 1.5s after the current node. This goal image is sent to ViNT along with the 4Hz image context to compute a short-range trajectory. This is tracked by a simple PID controller. The average progress towards the goal before collision is collected and reported across all trials. Table 3 summarizes the results of these experiments with multiple baselines and data sizes.

Adaptation Experiments

To test the new tasks, we adopt a similar evaluation setup to the fine-tuning experiments, but rely on the odometry position for the selected goal node rather than the image. For positional-adaptation, we move the goal coordinates into a local frame and send it to ViNT. For routing-adaptation, we determine the difference in lateral coordinates between the current node and the goal node. We choose the current node as reference to ensure an open-loop experiment and to allow for pre-computation of the command signals to be sent. We then apply the same binning strategy during training using a 0.05 normalized distance as the boundary between "left", "right", and "straight". The control system downstream of this is identical to image fine-tuning and the experiment terminates when at the goal or when colliding. The progress towards the goal before collision is collected and averaged across all trials in Table 3.

Baselines

We have the following baselines for the CARLA experiments:

1. **Scratch**: ViNT trained from scratch on the CARLA on-task dataset.
2. **Pre-trained Visual Representations**
   (a) **ImageNet**: ViNT initialized with the EfficientNet-B0 weights pre-trained on ImageNet, other parameters initialized from scratch, and fine-tuned with the CARLA on-task dataset.
   (b) **SimCLR**: ViNT initialized with the EfficientNet-B0 weights pre-trained with SimCLR [39] on the training data described in Section D.3, other parameters initialized from scratch, and fine-tuned with the CARLA on-task dataset.

(c) **VC-1**: ViNT initialized with a pre-trained ViT-B model checkpoint from the authors of VC-1 [167] *and frozen*, other parameters initialized from scratch, and fine-tuned with the CARLA on-task dataset. The VC-1 encoder is pre-trained on a combination of Ego4D, manipulation, navigation, and ImageNet images using Masked Auto-Encoding [94, 207].

3. **GNM**: The pre-trained embodiment-agnostic model checkpoint from the authors of GNM [234], fine-tuned with the CARLA on-task dataset. Note that GNM has 8.7M trainable parameters, compared to ViNT's 31M.

Method	Images	Positions
VC-1 [167]	0.19	0.49
ViNT-FE	0.32	0.78
ViNT	0.82	0.89

Table 26: Evaluation of ViNT fine-tuning with and without a frozen encoder, as compared to a general-purpose visual encoder. Even when frozen, ViNT's navigation-relevant features appear to transfer more readily to out-of-distribution inputs than general-purpose features.

We note that the VC-1 baseline's weak performance in Section 7.6.4 may be explained by the fact that it is *frozen*, while all other visual encoders were free to fine-tune. This is representative of typical downstream usage [167]. Despite training on multiple, diverse datasets, the visual representation's general-purpose features are not optimized for the navigation task, hampering zero-shot transfer to out-of-domain tasks. To provide a fair comparison of the quality of pre-trained visual features, we compare this performance to ViNT-FE (a pre-trained ViNT model that has it's visual encoder frozen). ViNT-FE has an equal number of trainable parameters to the VC-1 baseline, and frozen visual representations (see Table 26).

Hyperparameter	Value
ViNT Model	
# Parameters	31M
RGB Resolution	85×64
Encoder	EfficientNet-B0
Token Dimension	512
Attn. hidden dim.	2048
# Attention Layers n_L	4
# Attention Heads n_H	4
Temporal Context P	5
Prediction Horizon H	5
MLP layers	(256, 128, 64, 32)
ViNT Training	
# Epochs n_{ep}	30
Batch Size	300 [3]
Learning Rate	5×10^{-4}
Optimizer	AdamW [164]
Warmup Epochs	4
LR Schedule	Cosine
Scheduler Period	10
Compute Resources	8×V100 [3]
Training Time	30 hours [3]
Fine-tuning LR	1×10^{-4}
Diffusion Model	
# Parameters	318M
Resolution	128×128
# Up/Down Blocks	4
Attn. Resolutions	32, 16, 8
Layers per Block	2
Attn. Head Dim	8
Channels	(128, 128, 256, 512, 640)
Diffusion Type	continuous time
Noise Schedule	linear
Diffusion Training	
Dropout	0.1
Batch Size	128
Optimizer	AdamW
Warmup Steps	1000
Learning Rate	1e-4
LR Schedule	Cosine
Adam β_1	0.95
Adam β_2	0.999
Adam ϵ	1e-8
Weight Decay	0.001
EMA Inv. Gamma	1.0
EMA Power	0.75
EMA Max Decay	0.9999
CFG Mask Proportion	0.2
Train Steps	250,000
Training Time	30 hours
Compute Resources	v4-8 TPU board
Diffusion Sampling	
Sampler	DDIM [253]
DDIM η	0.0
Sampling Steps	200
Guidance Weight	1.0
Other	
Maximum distance	20
Distance tradeoff λ	0.01

Table 24: Hyperparameters for training ViNT and the diffusion model.

	Dataset	Platform	Speed	Hrs. Used	Environment
1	GoStanford	TurtleBot2	0.5m/s	14h	office
2	RECON	Jackal	1m/s	25h	off-road
3	CoryHall	RC Car	1.2m/s	2h	hallways
4	Berkeley	Jackal	2m/s	4h	suburban
5	SCAND-S	Spot	1.5m/s	4h	sidewalks
6	SCAND-J	Jackal	2m/s	1h	sidewalks
7	Seattle	Warthog	5m/s	1h	off-road
8	TartanDrive	ATV	10m/s	5h	off-road
9	NeBula	ATV	10m/s	10h	off-road
10	SACSoN	TurtleBot2	0.5m/s	10h	office
11	BDD	Car(s)	20m/s	4h	on-road
	Total			80h	

Table 25: The ViNT training dataset contains over 150 hours of navigation data in challenging indoor, outdoor, and off-road environments across 8 different robots of varying sizes, speeds, and capabilities.

APPENDIX E: NAVIGATION WITH FOUNDATION MODELS OF LANGUAGE, VISION, AND ACTION

E.1 PROMPT ENGINEERING

To use large language models for a particular task, as opposed to a general text completion, one needs to encode the task as a part of the text input to the model. There exist many ways to create such encoding and the process of the representation optimization is sometimes referred to as *prompt engineering* [205]. In this section, we discuss the prompts we used for **LLM** and **VLM**.

E.1.1 ***LLM** Prompt Engineering*

All our experiments use GPT-3 [22] as the **LLM**, accessible via OpenAI's API: https://openai.com/api/. We used this model to extract a list of landmarks from free-form instructions. The model outputs were very reliable and robust to small changes in the input prompts. For parsing simple queries, GPT-3 was surprisingly effective with a single, zero-shot prompt. See the example below, where the model output is highlighted:

> First, you need to find a stop sign. Then take left and right and continue until you reach a square with a tree. Continue first straight, then right, until you find a white truck. The final destination is a white building.
> Landmarks:
> 1. Stop sign
> 2. Square with a tree
> 3. White truck
> 4. White building

While this prompt is sufficient for simple instructions, more complex instructions require the model to reason about occurrences such as re-orderings, e.g. *Look for a glass building after after you pass by a white car*. We leverage GPT-3 ability to perform *in-context learning* [215] by adding three examples in the prompt:

> Look for a library, after taking a right turn next to a statue.
> Landmarks:
> 1. a statue
> 2. a library

```
Look for a statue. Then look for a library. Then go towards a pink house.
Landmarks:
1. a statue
2. a library
3. a pink house

[Instructions]
Landmarks:
1.🟩
```

We use the above prompt in all our experiments (Section 9.5.1, 9.5.2), and GPT-3 was successfully able to extract all landmarks. The comparison to other extraction methods is described in Section 9.5.3 and Appendix E.4.2.

E.1.2 *VLM Prompt Engineering*

In the case of our **VLM**— CLIP [205] — we use a simple family of prompts: *This is a photo of* ___, appended with the landmark description. This simple prompt was sufficient to detect over 95% of the landmarks encountered in our experiments. While our experiments did not require more careful prompt engineering, [205] and [299] report improved robustness by using an ensemble of slightly varying prompts.

E.2 BUILDING THE TOPOLOGICAL GRAPH WITH **VNM**

This section outlines finer details regarding how the topological graph is constructed using **VNM**. LM-Nav assumes access to observations from a prior traversal in the environment — for the experiments in our paper, we use a single human traversal followed by the graph generation process described below. Empirically, we found the system to be robust to the mechanism in which the traversal was collected (e.g. random, lawnmower, bee-lining), as long as the robot observes relevant landmarks at some point in the traversal.

We use a combination of learned distance estimates (from **VNM**), spatial proximity (from GPS), and temporal proximity (during data collection), to deduce edge connectivity. If the corresponding timestamps of two nodes are close ($< 2s$), suggesting that they were captured in quick succession, then the corresponding nodes are connected — adding edges that were physically traversed. If the **VNM** estimates of the images at two nodes are close, suggesting that they are *reachable*, then the corresponding nodes are also connected — adding edges between distant nodes along the same route and giving us a mechanism to connect nodes that were collected in different trajectories or at different times of day but correspond to the nearby locations. To avoid cases of underestimated distances by the model due to aliased observations, e.g. green open fields or a white wall, we filter out prospective edges that are significantly further away as per their GPS estimates — thus, if two nodes are nearby as per their GPS, e.g. nodes on different sides of a wall, they may not be disconnected if the **VNM** does not estimate a small distance; but two similar-looking nodes 100s of meters away, that may be facing a white wall, may have a small **VNM** estimate but are not added to the graph to avoid *wormholes*. alg:lmnav:orithm 14 summarizes this process — the timestamp threshold ϵ is 1 second, the learned distance threshold τ is 80 time steps (corresponding to ~ 20 meters), and the spatial threshold η is 100 meters.

Algorithm 14 Graph Building

```
Input: Nodes $n_i, n_j \in \mathcal{G}$ containing robot observations; VNM distance function $f_d$; hyperparameters $\{\tau, \epsilon, \eta\}$
Output: Boolean $e_{ij}$ corresponding to the existence of edge in $\mathcal{G}$, and its weight
learned distance $D_{ij} = f_d(n_i['image'], n_j['image'])$
timestamp distance $T_{ij} = |n_i['timestamp'] - n_j['timestamp']|$
spatial distance $X_{ij} = \|n_i['GPS'] - n_j['GPS']\|$
if $T_{ij} < \epsilon$ then
    return {True, $D_{ij}$}
else if $D_{ij} < \tau$ and $X_{ij} < \eta$ then
    return {True, $D_{ij}$}
else
    return False
end if
```

Since a graph obtained by such an analysis may be quite dense, we perform a *transitive reduction* operation on the graph to remove redundant edges.

E.3 Mobile Robot Platform

We implement LM-Nav on a Clearpath Jackal UGV platform (see Fig. 49(right)). The sensor suite consists of a 6-DoF IMU, a GPS unit for approximate localization, wheel encoders for local odometry, and front- and rear-facing RGB cameras with a 170° field-of-view for capturing visual observations and localization in the topological graph. The **LLM** and **VLM** queries are pre-computed on a remote workstation and the computed path is commanded to the robot wirelessly. The **VNM** runs on-board and only uses forward RGB images and unfiltered GPS measurements.

E.4 Miscellaneous Ablation Experiments

E.4.1 *Ablating the Search Objective*

The graph search objective described in Section 9.3.4 can be factored into two components: visiting the required landmarks (denoted by $P_l(\bar{v}|\bar{l})$) and minimizing distance traveled

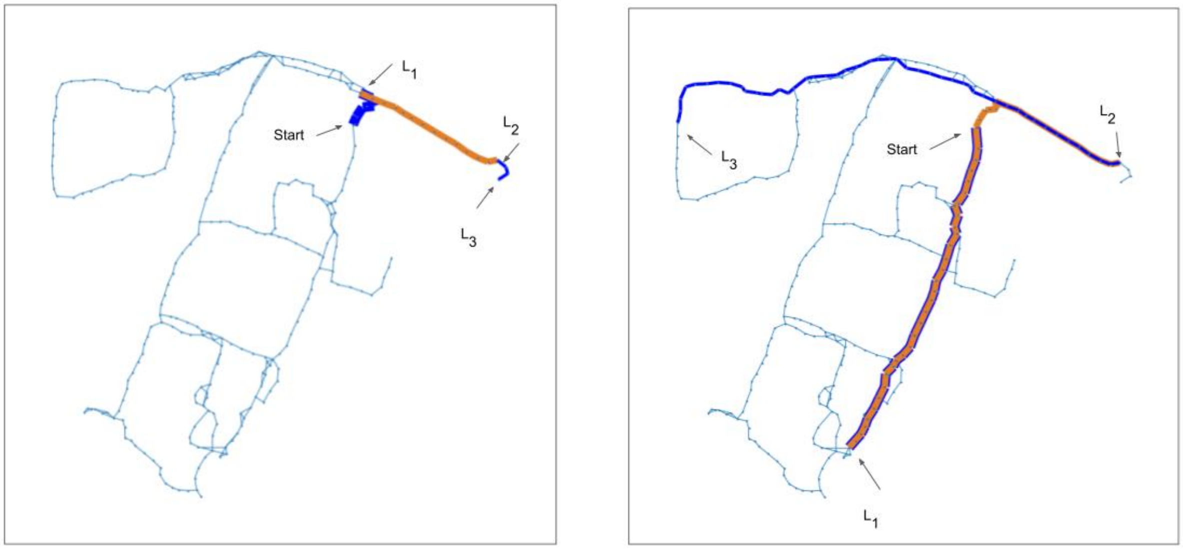


Figure 68: Examples of path planned by LM-Nav (left) and maximum likelihood planning (right). The start nodes and detected nodes are indicated with black arrows. In order to represent overlapping paths, we use colors interchangeably (start → L_1: blue, $L_1 \rightarrow L_2$: orange, $L_2 \rightarrow L_3$: blue). The path taken by LM-Nav is significantly shorter, resulting in a 5× more efficient plan.

(denoted by $P_t(\bar{v})$). To analyze the importance of these two components, we ran a set of experiments where the nodes to be visited are selected based only on P_l. This corresponds to a *Max Likelihood* planner, which only picks the most likely node for each landmark, without reasoning about their relative topological positions and traversability. This approach leads to a simpler alg:lmnav:orithm: for each of the landmark descriptions, the alg:lmnav:orithm selects the node with the highest CLIP score and connects it via the shortest path to the current node. The shortest path between each pair of nodes is computed using the Floyd–Warshall alg:lmnav:orithm.

Table 10 summarizes the performance metrics for the two planners. Unsurprisingly, the max likelihood planner suffers greatly in the form of efficiency, because it does not incentivize shorter paths (see Figure 68 for an example). Interestingly, the planning success suffers as well, especially in complex environments. Further analysis of these failure modes reveals cases where **VLM** returns erroneous detections for some landmarks, likely due to the contrastive objective struggling with variable binding (see Figure 69 for an example). While LM-Nav suffers from these failures as well, the second factor in the search objective $P_t(\bar{v})$ imposes a *soft constraint* on the search space of the landmarks, eliminating most of these cases and resulting in a significantly higher planning success rate.

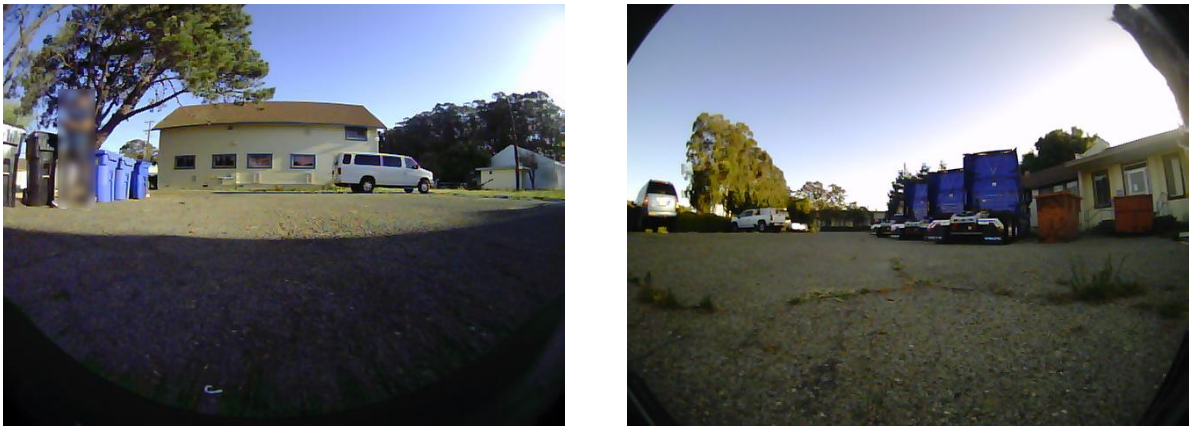


Figure 69: An example of failure to pick the correct image by maximum likelihood planning. Both images were selected for a prompt *A photo of a blue dumpster*. The left one was selected as a part of the LM-Nav's graph search and the right was selected by maximum likelihood planning. In the latter case, the selected image contains a blue semi-truck and an orange trailer, but no blue dumpsters. This might be an example of an issue with the variable binding. The left image was edited to maintain anonymity.

E.4.2 *Ablating the LLM*

As described in Section 9.5.3 we run experiments comparing performance of different methods on extracting landmarks. Here we provide more details on the experiments. The source code to run this experiments is available in the file ablation_text_to_landmark.ipynb in the repository.

As the **metric of performance** we used average extraction success. For a query with a ground truth list of landmarks L_{gt}, where a method extracts list L_m, we define the methods extraction success as:

$$\frac{|\mathrm{LCS}(L_m, L_{gt})|}{|L_{gt}|},$$

where LCS is longest common subsequence and $|\cdot|$ denotes a length of a sequence or a list. This metric is measuring not only if correct landmarks were extracted, but also whether they are in the same order as in the ground truth sequence. When comparing landmarks we ignore articles, as we don't expect them to have impact on the downstream tasks.

All the experiments were run using APIs serving models. We used OpenAI's API (https://beta.openai.com/) for GPT-3 and GooseAI (https://goose.ai) for the other open-source models. Both providers conveniently share the same API. We used the same, default parameters, apart from setting temperature to 0: we don't expect that landmark extractions to require creativity and model's determinism improves reputability. For all the reported experiments, we used the same prompt as described in Appendix E.1. Please check out the released code for the exact prompts used.

APPENDIX F: SEMANTIC GUESSWORK AS A HEURISTIC FOR PLANNING

F.1 IMPLEMENTATION DETAILS

F.1.1 *Hyperparameters*

	Parameter	Value
τ	Replanning Rate	1
δ	Language Influence Threshold	2m
n_s	Number of LLM Samples	10
w_p	Weight of Positive Scores	300
w_n	Weight of Negative Scores	150
	Max Time Steps	500

Table 27: Hyperparameters

F.1.2 *Computational Resources*

Parameter	Value
LLM	gpt-3.5-turbo[4]
Evaluation Runtime	5 hours
Compute Resources	$4 \times V100$
Total LLM Tokens	30M
Average API Cost	15 USD

Table 28: Parameters and resources required to run one evaluation round of LFG on the benchmark.

F.1.3 *Real World Results*

GENERATING PROMPTS: For both topological and geometric maps we use hand engineered methods for clustering objects in ways that the LLM can efficiently reason over. For geometric maps we implement two functions: *parseObjects* and *clusterObjects*. In our implementation, *parseObejcts* filters the geometric map and identifies the cluster centers among each class. *clusterObjects* takes the cluster centers and performs agglomerative clustering with a threshold of 6 meters, which is roughly the size of one section of a standard house. For topological maps we rely on the configuration of the four cameras to automatically perform parsing and clustering. In our implementation all the objects detected in each frame from either the front, left, right, or rear facing cameras is considered a single cluster.

PERCEPTION: For the hardware, we use a locobot base with a four HD logitech web cameras that are positioned at 90 degrees relative to each other. At each step of LFG each of four cameras is recorded and frames are semantically annotated. LFG directly uses these frames to determine if the robot should continue to move forward, turn left, turn right, or turn around a full 180 degrees. To improve the performance of our system we choose to whitelist a subset of the 20,000 classes. This reduces the size of the API calls to the language models and helps steer the LLM to focus on more useful information. Following is the complete whitelist used in our experiments:

- toaster
- projector
- chair
- kitchen table
- sink
- kitchen sink
- water faucet
- faucet
- microwave oven
- toaster oven
- oven
- coffee table
- coffee maker
- coffeepot
- dining table
- table
- bathtub
- bath towel
- urinal
- toilet
- toilet tissue
- refrigerator
- automatic washer
- washbasin
- dishwasher
- television set
- sofa
- sofa bed
- bed
- chandelier
- ottoman
- dresser
- curtain
- shower curtain
- trash can
- garbage

- cabinet
- file cabinet
- monitor (computer equipment)
- computer monitor
- computer keyboard
- laptop computer
- desk
- stool
- hand towel
- shampoo
- soap
- drawer
- pillow

LOW-LEVEL POLICY: The low-level policy running on the robot is the NoMaD goal-conditioned diffusion policy trained to avoid obstacles during exploration and determine which frontiers can be explored further [254].

HIGH-LEVEL PLANNING: For real-world experiments, we follow the setup of ViKiNG [232], where the agent runs a simple frontier-based exploration algorithm and incorporates the LLM scores as *goal-directed heuristics* to pick the best subgoal frontier. For simulation experiments, we use a geometric map coupled with frontier-based exploration, following the setup of [29]. Algorithms 15 and 16 summarize the high-level planning module in both cases.

Algorithm 15 Instantiating LFG with Topological Mapping

```
Input: $o_0$, Goal language query $q$
subgoal ← None
while not done do
    $o_t$ ← getObservation()
    frontierPoints ← mappingModule($o_t$)
    if $q$ in frontierPoints then
        turnTowardGoal(frontierPoints)
    else
        if numSteps % $\tau$ == 0 then
            location ← getCurrentLocation()
            $LLM_{pos}, LLM_{neg}$ ← scoreFrontiers(frontierPoints)
            scores ← []
            for point in frontier do
                distance ← distTo(location, point) scores[point] ← $w_p \cdot LLM_{pos}$ [$i$] - $w_n \cdot LLM_{neg}$ [$i$] - distance
            end for
            subgoal ← argmax(scores)
        end if
    end if
    numSteps ← numSteps +1
    goTo(subGoal)
end while
```

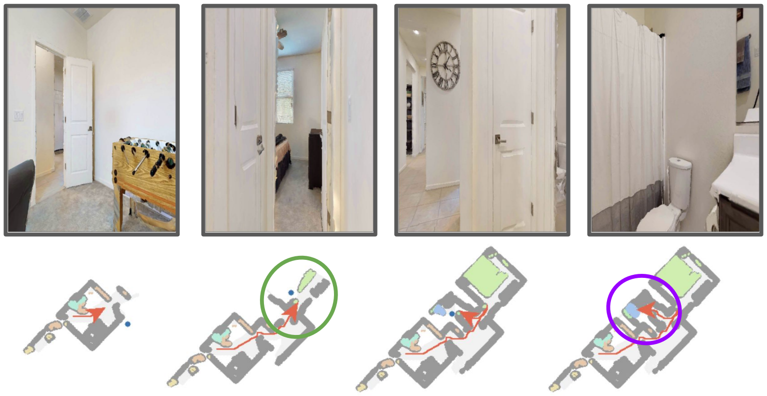
Algorithm 16 Instantiating LFG with Geometric Mapping

```
Input: $o_0$, Goal language query $q$
subgoal ← None
while not done do
    $o_t$ ← getObservation()
    obstacleMap, semanticMap ← mappingModule($o_t$[depth], $o_t$[semantic])
    if $q$ in semanticMap then
        subGoal ← getLocation(semanticMap, q)
    else
        if  numSteps % $\tau == 0$ then
            // replanning
            location ← getCurrentLocation()
            frontier ← getFrontier(obstacleMap)
            objects ← parseObjects(semanticMap)
            objectClusters ← clusterObjects(objects) $LLM_{pos}, LLM_{neg}$ ← ScoreSubgoals(objectClusters)

            scores ← []
            for point in frontier do
                distance ← distTo(location, point)
                scores[point] ← - distance
                closestCluster ← getClosestCluster(objectClusters, point)
                $i$ ← clusterID(closestCluster)
                if dist(closestCluster, point) $< \delta$ then
                    // incorporate language scores
                    scores[point] ← $w_p \cdot$ $LLM_{pos}$ [$i$] - $w_n \cdot LLM_{neg}$ [$i$] - distance
                end if
            end for
            subgoal ← argmax(scores)
        end if
    end if
    numSteps ← numSteps +1
    goTo(subgoal)
end while
```

F.1.4 *More Experiment Rollouts*

Figure 70 shows an example where the negative scoring is essential to LFG's success. Figures 71 and 72 show examples of LFG deployed in a previously unseen apartment and an office building, successfully exploring the environments to find an oven and a kitchen sink.



Query: Find the toilet.

1. LLM finds a bed, increases score to explore nearby.

2. No toilet found, LLM failure, FBE takes over.

3. FBE finds toilet by continuing exploration.

Agent succeeds!

Figure 70: Tolerance to LLM failures. An example rollout of LFG compensating for LLM failure. FBE takes over in this case and eventually succeeds, whereas the Greedy agent fails.

F.2 PROMPTS

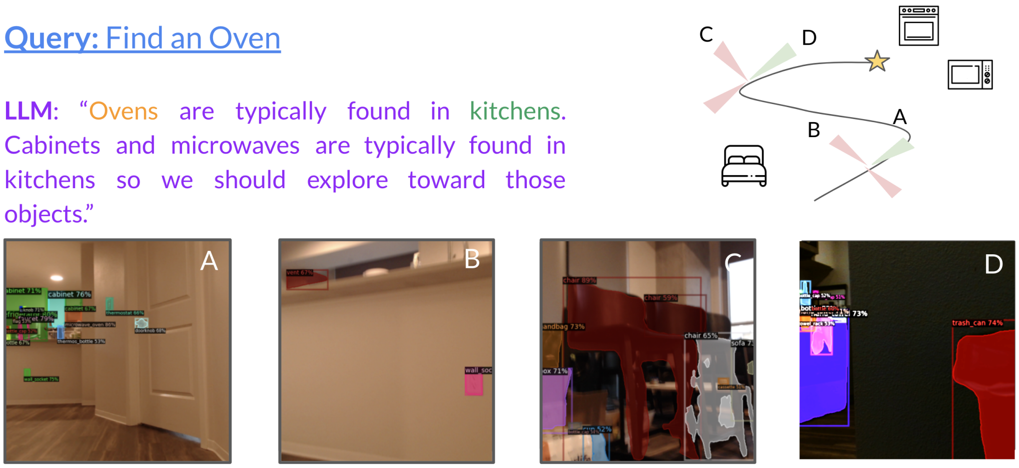
F.2.1 *Positive Prompt*

```
You are a robot exploring an environment for the first time. You
   will be given an object to look for and should provide guidance
    of where to explore based on a series of observations.
   Observations will be given as a list of object clusters
   numbered 1 to N.

Your job is to provide guidance about where we should explore next
   . For example if we are in a house and looking for a tv we
   should explore areas that typically have tv's such as bedrooms
   and living rooms.

You should always provide reasoning along with a number
   identifying where we should explore. If there are multiple
   right answers you should separate them with commas. Always
   include Reasoning: ¡your reasoning¿ and Answer: ¡your answer(s)
   ¿. If there are no suitable answers leave the space afters
   Answer: blank.

Example
```



Query: Find an Oven

LLM: "Ovens are typically found in kitchens. Cabinets and microwaves are typically found in kitchens so we should explore toward those objects."

Figure 71: LFG in an unseen apartment. The robot starts in the same starting location and environment as 59, and is tasked with finding an oven. LFG guides the robot towards the kitchen appliances, rather than the bedroom door, and successfully leads to the oven.

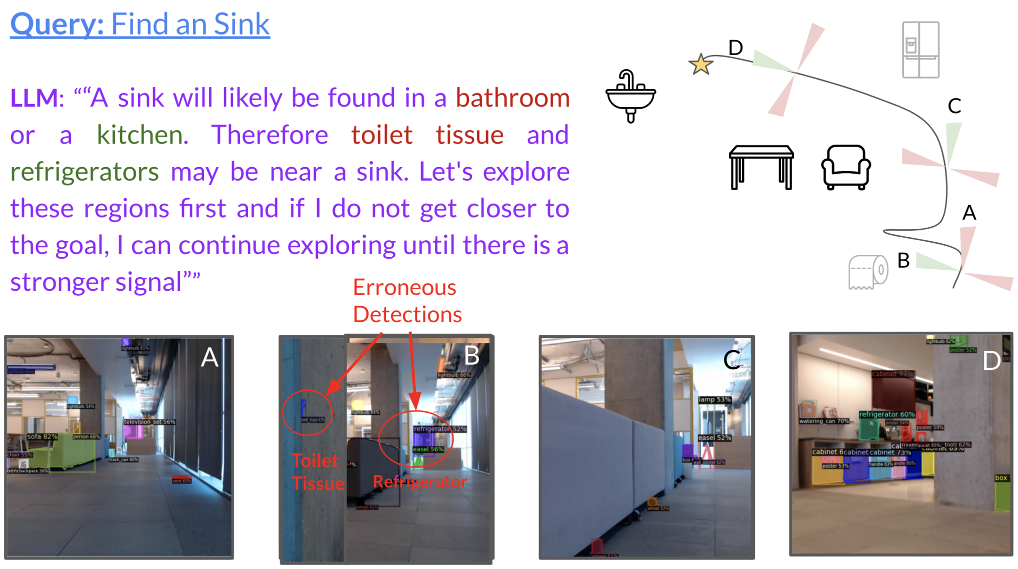
```
User:
I observe the following clusters of objects while exploring a
    house:

1. sofa, tv, speaker
2. desk, chair, computer
3. sink, microwave, refrigerator

Where should I search next if I am looking for a knife?

Assistant:
Reasoning: Knifes are typically kept in the kitchen and a sink,
    microwave, and refrigerator are commonly found in kitchens.
    Therefore we should check the cluster that is likely to be a
    kitchen first.
Answer: 3


Other considerations
```



Query: Find an Sink

LLM: ""A sink will likely be found in a bathroom or a kitchen. Therefore toilet tissue and refrigerators may be near a sink. Let's explore these regions first and if I do not get closer to the goal, I can continue exploring until there is a stronger signal""

Figure 72: LFG in an unseen office building. The agent looks for a sink in an open-plan office building. Despite erroneous detections, the robot continues exploring the environment, with LFG guiding it towards frontiers containing appliances found in a cafe. The robot successfully finds the sink despite imperfect detections.

```
1. Disregard the frequency of the objects listed on each line. If
   there are multiple of the same item in  a cluster it will only
   be listed once in that cluster.
2. You will only be given a list of common items found in the
   environment. You will not be given room labels. Use your best
   judgement when determining what room a cluster of objects is
   likely to belong to.
```

F.2.2 *Negative Prompt*

```
You are a robot exploring an environment for the first time. You
   will be given an object to look for and should provide guidance
    of where to explore based on a series of observations.
   Observations will be given as a list of object clusters
   numbered 1 to N.

Your job is to provide guidance about where we should not waste
   time exploring. For example if we are in a house and looking
   for a tv we should not waste time looking in the bathroom. It
   is your job to point this out.

You should always provide reasoning along with a number
   identifying where we should not explore. If there are multiple
   right answers you should separate them with commas. Always
   include Reasoning: ¡your reasoning¿ and Answer: ¡your answer(s)
   ¿. If there are no suitable answers leave the space afters
   Answer: blank.

Example

User:
I observe the following clusters of objects while exploring a
   house:

1. sofa, tv, speaker
2. desk, chair, computer
3. sink, microwave, refrigerator

Where should I avoid spending time searching if I am looking for a
    knife?

Assistant:
Reasoning: Knifes are typically not kept in a living room or
   office space which is what the objects in 1 and 2 suggest.
   Therefore you should avoid looking in 1 and 2.
Answer: 1,2


Other considerations
```

1. Disregard the frequency of the objects listed on each line. If there are multiple of the same item in a cluster it will only be listed once in that cluster.
2. You will only be given a list of common items found in the environment. You will not be given room labels. Use your best judgement when determining what room a cluster of objects is likely to belong to.