Towards Generally Intelligent Robots that Simply Work Everywhere

by

Nur Muhammad Shafiullah

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

August, 2025

Professor Lerrel Pinto

© Nur Muhammad Shafiullah

all rights reserved, 2025

To my grandmother, Shovona Khanom,

and to any who cross big oceans on small boats.

Acknowledgments

My journey did not start with my Ph.D., nor will it end here – and yet it is during my Ph.D. where I spent five of my most beautiful years in life. My unending gratitude to all the people in my life who made it so, without whose help reaching this point would be impossible.

First and foremost, I am grateful for my advisor, Lerrel Pinto. Being an amazing research advisor is the least of his accomplishments. In fact, Lerrel is one of the greatest human beings that I have met in my life. I had not done any robotics prior to this Ph.D., and yet, he took a chance in me – I would like to believe he did so because he saw and believed in my potential. And thus, I hope my work in this thesis honors his faith. I am also incredibly grateful to Lerrel for helping me survive and thrive during the sluggish COVID years. Starting a Ph.D. is difficult without having it coincide with a global pandemic, and yet Lerrel's constant support made it possible. I am proud of the projects we got to pursue together, and I am proud of the lab I got to help him build.

I must thank the members of my thesis committee: Rob Fergus, Saining Xie, Charlie Kemp, and Russ Tedrake, who have been my great supporters and mentors throughout the years. Rob's encouragement and support throughout my Ph.D. journey specifically made some heavy burdens light. Saining is a constant source of inspiration in thinking out of the box. Charlie's contribution to robots in homes is undeniable – and without his work on Stretch this thesis could not exist. Finally, Russ has been a guiding light – not just for me but for the entire field – on how we can keep asking hard scientific questions in this attention economy.

My heartfelt gratitude to my research mentor and collaborators – Soumith Chintala, Chris Paxton, and Arthur Szlam. Soumith has been almost a second advisor to me – and his advice about adoption and community driven open sourcing is something I will carry with me for a long time. I admire Chris' passion about all things robotic, and I love Arthur's way of thinking outside the box about robot memory.

I am grateful for the students I got to mentor and collaborate with during my time at NYU. Haritheja, Jeff, Peiqi, Anant, Jyo are equal parts my collaborators, mentees, and inspirations. But even beyond that, Siddhant, Ulyana, Nikhil, Kathy, Irmak, Sridhar, Abitha, Sneha, Jay, Venky, Aadhi, Enes, Yaswanth, and many many more who are slipping my mind right now have been incredible friends and peers in this long journey. I hope I have been someone you can look up to.

My thanks also goes to those ahead of me in this journey at NYU – Mark, Aahlad, Raghav, Mimee, Karl, David, Denis, Ilya, Roberta, Raunaq, and many more. You have been friends, inspirations, and often, someone I could turn to with questions about life or research.

A special set of thanks go to the heroes behind the scenes – Hong always with administrative answers, Shenglong with all my compute cluster questions, or Santiago with the academic planning. I hope you know how much I appreciate your hard work.

Another special thanks goes to the team of Hello Robot – a large part of my thesis research was possible because they created such a light and friendly platform for robotics. I would like to express my gratitude to Aaron, Charlie, Blaine, Binit, and the rest of the team for their pioneering steps of robots in homes.

I am more than my research – I try to be a complete and well-rounded human being. To get to where I am, the contributions of the following people have been critical.

My heartiest gratitude to my grandmother, Shovona Khanam. She told me stories of greatness, bestowed upon me my dreams, and led me to believe that even a kid from a middle-class family in Bangladesh could someday do something worth remembering. I regret that she passed away on

November 18, 2018, before she could see me realize my dreams, but I hope she knows that her dreams survive within her grandchildren.

Thanks to my parents in Bangladesh, my mother Arifa Parvin Khan, and my father Md Matiur Rahman Mollah. They acquiesced to their youngest child leaving them for an unknown land seven thousand and seven hundred miles away, even when it broke their hearts.

Thanks to my brother, Md Mohsinur Rahman Adnan. He holds home in his heart and loves me through whatever I do from then on out.

Thanks to Pranon Rahman Khan, old friend who is not here anymore. He recommended me more than two hundred books, and through them made me the person I am today.

Thanks to all of my friends in Next House 3E and MIT Bangladeshi "mafia", and others too many to list. They welcomed me with open arms and never let me feel away from home.

Thanks to my friends in Bangladesh, who saw me a lot in my early year, and then not at all, and yet kept supporting me throughout. Specially Hasan Shahriar Jisan, who has been a friend, confidant, and a supporter for almost the last 18 years.

Thanks to the entire Olympiad infrastructure in Bangladesh, which brought me outside of Bangladesh for the first time. Special thanks to Dr. Mahbub A. Majumdar, and my mentors, Haque Muhammad Ishfaq and Tarik Adnan Moom.

Thanks to my friends who kept me sane through the COVID years – Sanzeed Anwar, Lauren Huang, and John Gu. I hope you won't have to do things yourselves because robots will do it better.

My final heartfelt thanks goes to my partner Alexa Gross. You are the kindest and most genuine person in my life. Thanks for helping me navigate the most difficult parts of my Ph.D. My thanks also goes to Alexa's family, including Steve, Meiyan, and Kodiak Gross, and her mentor Kathy Caraccio. Your love keeps me afloat.

Abstract

Applications of machine learning have touched the lives of common people in innumerable novel ways. Robotics today seems poised to make such an impact, too. Yet the current state-of-the-art in robotics, whether it's a parkouring humanoid from Boston Dynamics or a T-shirt-folding robot from Google Deepmind, are specialists of their own environments – either by instrumenting and extensively modeling the scene, or by collecting weeks or months of data on the exact same setup.

In this thesis, we focus on *building generally intelligent robots that simply work everywhere* by studying the interplay of representation, data, and memory in robotics. To create robots that can address the broad and diverse challenges of operating in messy and unstructured environments *everywhere*, this thesis investigates three fundamental directions. We first look into algorithms that optimize the use of data in robot learning since data, as fuel, plays a critical role in creating broadly capable ML systems. We not only create efficient, self-supervised representations of the robots' perception, but also develop action representations that enables scaling to large, uncurated demonstration datasets. Then, we take a deep dive on creating systems – bridging algorithms and hardware – that can create and learn from robot data in the wild. Such systems enable few-shot and then zero-shot behavior generalization in novel homes in New York City and beyond. Finally, to enable generally intelligent robot behavior that extends over time and space, we construct neural data structures called *spatio-semantic memory* for robots. These memory modules enable scaling in-the-wild autonomous robot behavior from seconds to hours, and beyond.

Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

Over the last five years, one of the defining feature of machine learning has been its ability to touch the everyday lives of people. Today, more than any time in the past, people are using large language models and image generation models in their chat apps, and seeing self-driving cars in their street. It is reasonable, therefore, to expect to see a similar breakthrough moment for robotics – bringing generally intelligent robots that can assist us everywhere and in almost every task.

In such a moment where large ML models are demonstrating remarkable capabilities in digital domains, from generating mesmerizing scenery to solving calculus problems with near-human accuracy, it is natural to ask – where are the physical equivalents? The almost-magic spell of introducing data and learning falls short of bringing us robot butlers in every home, even today, because of a fundamental assumption core to all machine learning. Assuming that the training and test domains are congruent underlies each landmark achievements in machine learning, at least in digital domains, and yet it is not so self-evident in robotics. Therefore, the state-of-the-art robots from research labs are experts of many tasks but are constrained within that same lab in the same scene, or they are narrow-domain experts, like roombas or dishwashers. Marrying the two to build general robots that can navigate unstructured physical tasks in messy environments therefore requires solutions that can somehow circumvent this common, core assumption of learning.

This thesis focuses on finding *practical* solutions to this problem, building towards generally intelligent robots that simply generalize to novel environments and scenes out of the box, by

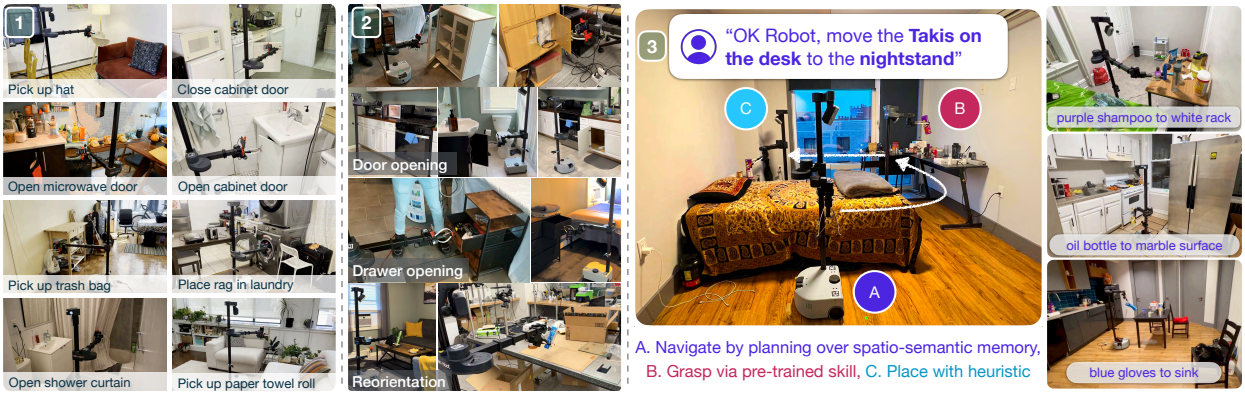


Figure 1.1: Interplay of representation, data, and memory in robotics enables robots in arbitrary homes and live demo environments. (1) Self-supervised visual representation learning [Pari et al. 2021] unlocks few-shot skill learning from 5 mins. of data and 15 mins. of fine-tuning [Shafiullah et al. 2023b]. (2) Multi-modal behavior cloning [Shafiullah et al. 2022; Cui et al. 2022; Lee et al. 2024] can train policies on diverse data that generalize to novel scenes and objects zero-shot [Etukuru et al. 2024]. (3) Semantic memory [Shafiullah et al. 2023a; Liu et al. 2024b] allows long-horizon, zero-shot operation in arbitrary open-world scenes [Liu et al. 2024c].

addressing a trifecta of roadblocks. The first bottleneck we focus on is the scalability of supervised robot policy learning through behavior cloning – both on how to enable models to learn from the most amount of available data, and how to extract the most amount of useful information from the limited data we may get in the wild. Our solution to this focuses on the representations for policy learning – creating efficient self-supervised visual representations, and continuous-discrete hybrid action representations that unlock learning from multi-modal, uncurated behavior data. The second is the availability of systems that can efficiently generate and use robot behavior data in the wild. We take a holistic approach to this, designing algorithms and hardware that, combined together, enable progressively few-shot and zero-shot generalizable behavior in messy and unstructured environments. The final piece of the puzzle we address is in long-horizon autonomous robot behavior – how a robot should represent its (unstructured) environment over long periods of interactions from both itself and other humans or robots. Our answer lies in spatio-semantic representations that takes advantage of advances in large vision and language models to build an environment representation zero-shot.

Before going into the main contributions of this thesis in more detail, this introductory section will present some background and motivation for each of the problems mentioned above.

1.1 Learning Representations for Scalable Policy Learning

In this thesis, we are primarily concerned with learning robotic behavior in a supervised fashion, primarily through behavior cloning. Contrast this with learning robotic behavior in an unsupervised manner using Reinforcement Learning [Sutton and Barto 2018], which we will not cover here but mention future possibilities in Chapter 12.

To be concrete and rigorous, let us define the problem of behavior cloning from a robotics perspective. The behavior cloning problem is defined by a dataset $\mathcal{D}$, that contains pairs of samples (o, a): an observation $o \in \mathcal{O}$ and an action $a \in \mathcal{A}$, and the objective is to learn a function $\pi : \mathcal{O} \rightarrow \mathcal{A}$. Typically, the observation o contains the sensory information of a robot, which can include data from robot- or environment-mounted cameras, robot proprioception information, or other sensory information such as tactile, force, torque information etc. The action a typically denotes some physical change the robot is able to manifest and can be defined in a variety of spaces – for example by changing its own joint angles, velocities, or torques, or on a higher, more abstract space, by moving its end-effectors in a certain way in cartesian space. Typically, we assume that the dataset $\mathcal{D}$ is created and curated in a way such that it contains samples from the behaviors of an "optimal" agent.

Given these, in practice, the problem of behavior cloning or supervised policy learning comes down to learning the function π that also exhibits the same optimal behavior when deployed in the real world environment. In practice, we use deep neural networks to approximate π as π_θ, so our challenge comes down to defining a learning algorithm $\mathcal{L}$ such that the function π_θ has the desired properties. There are practical issues of learning online behavior from offline datasets

[Ross et al. 2011], but our primary inquiry in this work will be about improving the scalability of such algorithms. We define scalability of a policy learning algorithm in two major axes:

- Even when $\mathcal{D}$ is small, the algorithm $\mathcal{L}$ is able to extract useful priors out of it,
- As $\mathcal{D}$ grows larger, the algorithm $\mathcal{L}$ is able to take advantage of the more and diverse dataset.

We accomplish the first goal by designing a self-supervised learning algorithm for the robotic observation (Chapter 2, following Pari et al. [2021]), and the second goal by creating a hybrid representation for actions (Chapter 3- 5 which follows Shafiullah et al. [2022]; Cui et al. [2022] and Lee et al. [2024] respectively). Note that each chapter may consider slight variations of these and related settings, but we will be explicit about the specific setting and notation considered within each chapter.

1.2 Mechanisms for Generalizable Scaling In-the-wild

While improving policy learning algorithms can help create better robot behavior given a dataset, often that is not the only bottleneck in creating robust and generalizable behaviors for robots in the wild. Rather, the primary bottleneck comes down to the assumption of supervised learning: that there *already exists* a dataset with the desirable properties of optimal behavior.

Compare this with the reality of deploying robots in the wild. Unlike vision, language, or even biological models, we do not have existing large scale datasets or a full internet repertoire that we can simply scrape. First, with teleoperation, the most common method of collecting robot data today, collecting data at scale is labor-intensive and incredibly expensive – making it nearly impossible in academic settings. Secondly, even with a large budget, the scaled datasets still exist within a limited set of environments [Team et al. 2025a,b]. Recall, however, that the core

assumption of ML discussed previously requires scaling up data in the wild if the resultant robots are to be deployed in the wild. Therefore, the key question that we try to answer in this part of the thesis is:

> *How can we scale up robot data in the wild that is useful for learning policies without having to spend an immense amount of resources on it?*

Each of the factors mentioned above can be a critical problem and has plagued prior works attempting in the wild data collection. For example, while Khazatsky et al. [2024] undertook an impressive, multi-university collaboration to collect data with a robot setup, because of the high resource demands iteration on the data collection and policy training process was not possible. Similarly, approaches using Augmented Reality methods for collecting data falls short of capturing the contact-rich interaction with the physical world. All of these difficulties make the data collected with such processes relatively less useful for learning large scale, generalizable policies.

In this process, it is important to introduce the concept of *robot-free robot data collection*. While classically, teleoperating an entire robot has been the process through which robot datasets have been collected, it may not be entirely necessary. Robot-free data collection emphasizes collecting data that matches the observation o and the action a of the robots, while abstracting away the robot infrastructure a way that makes it operationally easier – for example, by using a kinematically equivalent mechanism [Wu et al. 2023] or a handheld tool with a camera [Song et al. 2020]. Compared to teleoperated data collection methods, however, robot free data collection methods generally tend to be more holistic – since the mapping from the system to the robot is not always exactly one-to-one, often more algorithmic work is required to close that gap. Therefore, such methods are usually presented as a system – combined with data collection methods and effective policy learning algorithms.

In this thesis, we will discuss some works in this direction of scaling robot data with robot-free data collection by introducing a sequence of systems. In Chapter 6 [Shafiullah et al. 2023b], we

will introduce a preliminary system which can start scaling data in the wild and show few-shot performance with a self-supervised representation learning inspired algorithm. We expand upon it in Chapter 7 [Etukuru et al. 2024], showing that data collected with these handheld tools, when pooled and combined with a proper learning algorithms, can perform similar tasks zero-shot in the wild. Finally, in Chapter 8, we open the way for such work to expand beyond a single arm and static policies. In all of these works, there will be further details of the physical hardware that is used to scale the dataset and deploy the policies, as well as the algorithm that is used to train the policies that generalize in-the-wild.

1.3 Robotic Memory for Long-horizon Intelligent Behavior

Most of what is discussed from a policy learning perspective in today's robot learning literature usually concern themselves with a static and almost always tabletop setup. However, the world is much larger than just tabletops, and even most of mobile robotics today focuses on locomotion and not manipulation. To address this gap, this section of the thesis will focus on *mobile manipulation* and long-horizon challenges.

We generally call a problem mobile manipulation if, in order to complete the manipulation task, the robot is required move *itself* from one location to another. There are a few further assumptions generally made about mobile manipulation that makes it more challenging than static manipulation. First, the observation sensors e.g. the cameras are all assumed to be installed on the robot – therefore giving the robot a partial view of the environment. Second, the robot is expected to not only manipulate but also navigate well – so completing the task while knocking over task-irrelevant objects may still be considered a failure. Finally, empirically, the robot is evaluated on more temporally extended objectives compared to static manipulation, which makes succeeding at the task harder. Due to these assumptions, a few challenges compound. For example, partial observability means that a purely reactive policy is less likely to succeed in a mobile

manipulation setting – the robot needs to synthesize information about the world before acting on it. Or, because of its temporally extended nature, collecting teleoperated data for a mobile manipulation task becomes harder, and therefore, end-to-end policies become harder to manage.

In this work, we therefore focus on the problem of creating a *robotic spatio-semantic memory* that addresses a lot of these challenges. Our work follows the footsteps of earlier planning-based robotics work, where a robotic agent creates a map of an environment and then plans its task over it. We define a robotic spatio-semantic memory as a data structure $\omega \in \Omega$, where Ω is the space of maps, and $\tau \in \mathcal{T}$ are potential semantic queries, capable of at least the following:

- **Ingest** ($\Omega \times \mathcal{O} \rightarrow \Omega$): Given a new observation o from the robot, ω is able to update its inner representation of the environment into an updated map, ω'.
- **Spatio-semantic Query** ($\Omega \times \mathcal{T} \rightarrow \mathcal{R}^3$): The map ω should be able to respond to a semantic query τ with a relevant (x, y, z) location in 3-D space.

While the ingestion for semantic maps look similar to most semantic robot memories, there can be many variations in the query method. Namely, some memory structures may return additional information, such as a pose as well as a position, or a confidence interval on the response, alongside the query response. But in its most bare-bone form, the spatio-semantic memory structures should be able to respond to the basic form of the query.

Such a spatial memory data structure readily responds to a lot of challenges of mobile robotics. First, the memory handles the need to integrate partial observations of the environment over time into a consistent representation on which the robot's behavior policy can operate. The ingestion mechanism, if designed well, can help the robot deal with long-horizon challenges by operating on the memory ω rather than the observation o. Second, a properly designed memory is able to provide further affordance beyond what is required for manipulation – such as a safe navigation path or a collison avoidant trajectory.

In this thesis, we present spatio-semantic memory algorithms created with the explicit goal of being open-world and open-vocabulary capable. Open-world here means that the model should not make prior assumptions about the structure of the environment or the existence of fiducial markers in the scene – it should be able to handle arbitrary everyday environments. Open-vocabulary means that the memory should be able to respond, or at least extend to, to queries from an arbitrarily large dictionar – it should not rely on mapping only a closed set of objects or categories defined prior to runtime. The works presented in this thesis will show how we can distill pre-trained large vision language models zero shot to create such spatial scene memories in Chapter 9 [Shafiullah et al. 2023a]. Then, in Chapter 10 [Liu et al. 2024c], we will show how we can combine pre-trained manipulation policies into this to create mobile manipulation policies that generalize to novel environments. Finally, Chapter 11 [Liu et al. 2024b] will show how these memories can be updated online and dynamically as the world changes around the robot due to its own actions or that of humans.

1.4 Some Words about Evaluation

While creating robot policies and systems that generalize to the world is hard, it is even harder to properly evaluate such systems. A lot of current approaches are quite ad-hoc, especially in generalizable robotics, with a Laissez-faire attitude of "you will know it when you see it". When it is done properly, the time, money, and labor required to do the necessary evaluations in the real world can be expensive [Team et al. 2025b] and out of reach for academic researchers.

In all of our experiments in this thesis, we try to maintain a high bar by evaluating our robot policies and systems in the real world, and whenever possible, in messy, unstructured environments in the wild. We make the best possible recommendation given experimental evidences, not only using a single set of experiments with a hard-to-find statistical soundness, but with intuition built up over years of deploying real robots in real homes of New York city and other parts of USA. The goal is

to create generally intelligent robots that just works everywhere, and this thesis aims to bring some clarity in understanding what is important towards that end.

Part I

Representations for Perception and Control

2 | Surprising Effectiveness of Representation Learning for Behavior Cloning: Visual Imitation with Nearest Neighbors

2.1 Introduction

Imitation learning serves as a powerful framework for getting robots to learn complex skills in visually rich environments [Zhang et al. 2018b; Stadie et al. 2017; Duan et al. 2017; Zhu et al. 2018; Young et al. 2020]. Recent works in this area have shown promising results in generalization to previously unseen environments for robotic tasks such as pick and place, pushing, and rearrangement [Young et al. 2020]. However, such generalization is often too narrow to be directly applied in the diverse real-world application. For instance, policies trained to open one door rarely generalize to opening different doors [Urakami et al. 2019]. This lack of generalization is further exacerbated by the plethora of different options to achieve generalization: either needing hundreds of diverse demonstrations, task-specific priors, or large parametric models. This begs the question: What really matters for generalization in visual imitation?

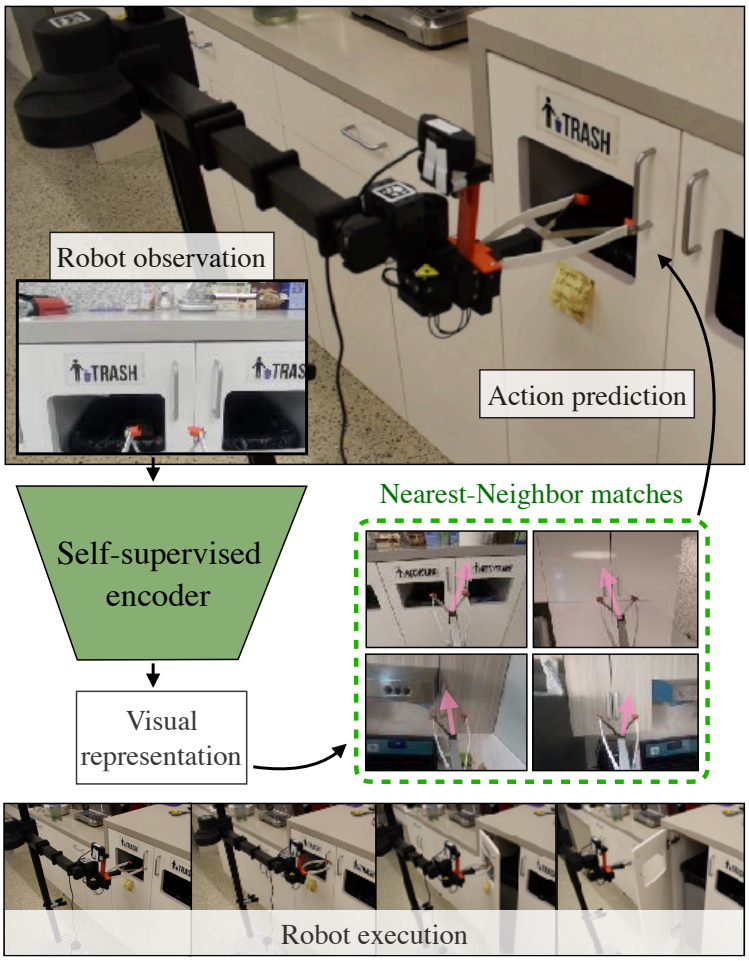


Figure 2.1: Consider the task of opening doors from visual observations. VINN, our visual imitation framework first learns visual representations through self-supervised learning. Given these representations, non-parametric weighted nearest neighbors from a handful of demonstrations is used to compute actions, which results in robust door-opening behavior.

An obvious answer is visual representation – generalizing to diverse visual environments should require powerful representation learning. Prior work in computer vision [Grill et al. 2020; Chen et al. 2020c,d; Caron et al. 2020; Bardes et al. 2021] have shown that better representations significantly improve downstream performance for tasks such as image classification. However, in the case of robotics, evaluating the performance of visual representations is quite complicated. Consider behavior cloning [Torabi et al. 2018], one of the simplest methods of imitation. Standard approaches in behavior cloning fit convolutional neural networks on a large dataset of expert

demonstrations using end-to-end gradient descent. Although powerful, such models conflate two fundamental problems in visual imitation: (a) representation learning, i.e. inferring information-preserving low-dimensional embeddings from high-dimensional observations and (b) behavior learning, i.e. generating actions given representations of the environment state. This joint learning often results in large dataset requirements for such techniques.

One way to achieve this decoupling is to use representation modules pre-trained through standard proxy tasks such as image classification, detection, or segmentation [Sax et al. 2019]. However, this relies on large amounts of labelled human data on datasets that are often significantly out of distribution to robot data [Chen et al. 2020a]. A more scalable approach is to take inspiration from recent work in computer vision, where visual encoders are trained using self-supervised losses [Chen et al. 2020d,c; Grill et al. 2020]. These methods allow the encoders to learn useful features of the world without requiring human labelling. There has been recent progress in vision-based Reinforcement Learning (RL) that improves performance by creating this explicit decoupling [Stooke et al. 2021; Yarats et al. 2021b]. Visual imitation has a significant advantage over RL settings: learning visual representations in RL is further coupled with challenges in exploration [Yarats et al. 2021a], which has limited its application in real-world settings due to poor sample complexity.

In this work we present a new and simple framework for visual imitation that decouples representation learning from behavior learning. First, given an offline dataset of experience, we train visual encoders that can embed high-dimensional visual observations to low-dimensional representations. Next, given a handful of demonstrations, for a new observation, we find its associated nearest neighbors in the representation space. For our agent's behavior on that new observation, we use a weighted average of the nearest neighbors' actions. This technique is inspired by Locally Weighted Regression [Atkeson et al. 1997], where instead of operating on state estimates, we operate on self-supervised visual representations. Intuitively, this allows the behavior to roughly

correspond to a Mixture-of-Experts model trained on the visual demonstrations. Since nearest neighbors is non-parametric, this technique requires no additional training for behavior learning. We will refer to our framework as Visual Imitation through Nearest Neighbors (VINN).

Our experimental analysis demonstrates that VINN can successfully learn powerful representations and behaviors across three manipulation tasks: Pushing, Stacking, and Door Opening. Surprisingly, we find that non-parametric behavior learning on top of learned representations is competitive with end-to-end behavior cloning methods. On offline MSE metrics, we report results on par with competitive baselines, while being significantly simpler. To further test the real-world applicability of VINN, we run robot experiments on opening doors using 71 visual demonstrations. Across a suite of generalization experiments, VINN succeeds 80% on doors present in the demonstration dataset and 40% on opening the door in novel scenes. In contrast, our strongest baselines have success rates of 53.3% and 3.3% respectively.

To summarize, this paper presents the following contributions. First, we present VINN, a novel yet simple to implement visual imitation framework that derives non-parametric behaviors from learned visual representations. Second, we show that VINN is competitive to standard parametric behavior cloning and can outperform it on a suite of manipulation tasks. Third, we demonstrate that VINN can be used on real robots for opening doors and can achieve high generalization performance on novel doors. Finally, we extensively ablate over and analyze different representations, amount of training data, and other hyperparameters to demonstrate the robustness of VINN.

2.2 Related Work

2.2.1 Imitation via Cloning

Imitation learning is frequently used to learn skills and behaviors from human demonstrations [Piaget 2013; Meltzoff and Moore 1977, 1983; Tomasello et al. 1993]. In the context of manipulation, such techniques have successfully solved a variety of problems in pushing, stacking, and grasping [Zhang et al. 2018b; Zhu et al. 2018; Argall et al. 2009; Hussein et al. 2017]. Behavioral Cloning (BC) [Torabi et al. 2018] is one of the most common techniques. If the agent's morphology or viewpoint is different than the demonstrations', the model needs to involve techniques such as transfer learning to resolve this domain gap [Stadie et al. 2017; Sermanet et al. 2016]. To close this unintended domain gap, [Zhang et al. 2018b] has used tele-operation methods, while [Song et al. 2020; Young et al. 2020] have used assistive tools. Using assistive tools provides us the benefit of being a able to scalably collect diverse demonstrations. In this paper, we follow the DemoAT [Young et al. 2020] framework to collect expert demonstrations.

2.2.2 Visual Representation Learning

In computer vision, interest in learning a good representation has been longstanding, especially when labelled data is rare or difficult to collect [Chen et al. 2020c,d; Grill et al. 2020; Caron et al. 2020]. This large class of representation learning techniques aim to extract features that can help other models improve their performance in some downstream learning tasks, without needing to explicitly learn a label. In such tasks, first a model is trained on one or more pretext tasks with this unlabeled dataset to learn a representation. Such tasks generally include instance invariance, or predicting some image transformation parameters (e.g. rotation and distortion), patches, or frame sequence [Gidaris et al. 2018; Dosovitskiy et al. 2015; Doersch et al. 2016; Misra et al. 2016;

Chen et al. 2020c,d; Wu et al. 2018]. In representation learning, the performance of the model on the pretext task is usually disregarded. Instead, the focus is on the input domain to representation mapping that these models have learned. Ideally, to solve such pretext tasks, the pretrained model may have learned some useful structural meaning and encoded it in the representation. Thus, intuitively, such a model can be used in downstream tasks where there is not enough data to learn this structural meaning directly from the available task-relevant data. Unsupervised representation learning, in works such as [Chen et al. 2020c,d; Grill et al. 2020; Caron et al. 2020; Bardes et al. 2021; Dwibedi et al. 2021], has shown impressive performance gains on difficult benchmarks since they can harness a large amounts of unlabelled data unavailable in task-specific datasets.

Recently, interest in unsupervised or semi-supervised representation learning technique has grown within robotics [Manuelli et al. 2020] due to the availability of unlabeled data and its effectiveness in visual imitation tasks [Young et al. 2021; Zhan et al. 2020]. We follow a BYOL-style [Grill et al. 2020] self-supervised representation learning framework in our experiments.

2.2.3 Non-parametric Control

Non-parametric models are those, which instead of modeling some parameters about the data distribution, tries to express it in terms of previously observed training data. Non-parametric models are significantly more expressive, but as a downside to this, they usually require a large number of training examples to generalize well. A popular and simple example of non-parametric models is Locally Weighted Learning (LWL) [Atkeson et al. 1997]. LWL is a form of instance-based, non-parametric learning that refers to algorithms whose response to any query is a weighted aggregate of similar examples. Simple nearest neighbor models are an example of such learning, where all weight is put on the closest neighbor to the input point. Nearest neighbor methods have been successfully used in previous works for control tasks [Mansimov and Cho 2018] More sophisticated, k-NN algorithms base their predictions on an aggregate of the nearest k points

[Aha and Salzberg 1994].

Uses of LWL based methods in supervised learning, robotics, and reinforcement learning is quite old. In works like [Snell et al. 2017; Wang et al. 2019], effectiveness of LWL algorithms like k-nearest neighbor has shown competitive success in difficult, high dimensional tasks like classifying the miniImageNet. LWL has also shown success for robotic control problems [Atkeson et al. 1997], although it requires an accurate state-estimator to obtain low-dimensional states. In [Lee and Anderson 2016; Pritzel et al. 2017; Rajeswaran et al. 2018], elements of non-parametric learning is weaved into the reinforcement learning algorithms to create models which can adjust their complexity based on the amount of available data. Finally, in works like [Shah and Xie 2018] non-parametric k-Nearest Neighbor regression based Q-functions are shown to give a good approximation of the true Q function under some theoretical assumptions. Our work, VINN, draws inspiration from the simplicity of LWL and demonstrates the usefulness of this idea by using Locally Weighted Regression in challenging visual robotic tasks.

2.3 Approach

In this section, we describe the components of our algorithms and how they fit together to create VINN. As seen in Fig. 2.2, VINN consists of two parts: (a) training an encoding network on offline visual data, and (b) querying against the provided demonstrations for a nearest-neighbor based action prediction.

2.3.1 Visual Representation Learning

Given an offline dataset of visual experience from the robot, we first learn a visual representation embedding function. In this work, we use two key insights for learning our visual representation: first, we can learn a good vision prior using existing large but unrelated real world datasets, and

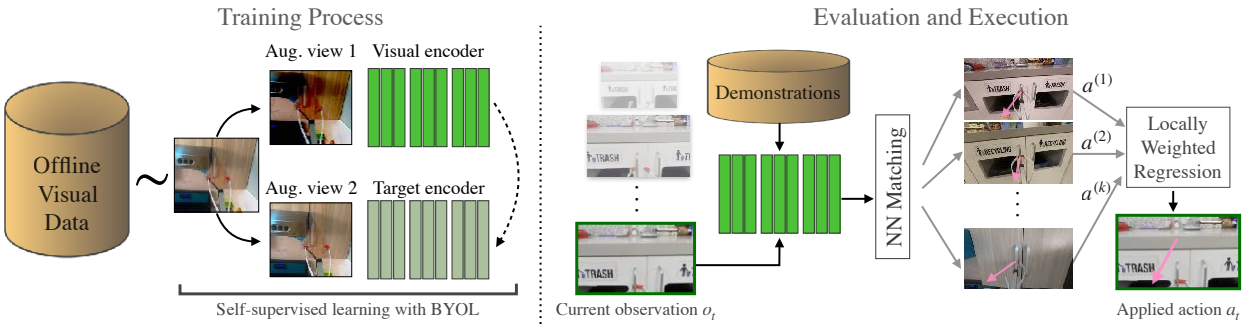


Figure 2.2: Overview of our VINN algorithm. During training, we use offline visual data to train a BYOL-style self-supervised model as our encoder. During evaluation, we compare the encoded input against the encodings of our demonstration frames to find the nearest examples to our query. Then, our model's predicted action is just a weighted average of the associated actions from the nearest images.

then, we can fine-tune starting from that prior using our demonstration dataset, which is small but relevant to the task at hand.

For the first insight, whenever possible, we initialize our models from an ImageNet-pretrained model. Such models are provided with the PyTorch [Paszke et al. 2019] library that we use and can be achieved by simply adding a single parameter to the model initialization function call.

Then, we use self supervised learning and train this visual encoder on the all the frames in our offline training dataset. In this work, we use Bootstrap Your Own Latent (BYOL) [Grill et al. 2020] as the self-supervision objective. As illustrated in Fig. 2.2, BYOL uses two versions of the same encoder network: one normally updating online network, and a slow moving average of the online network called the target network. The BYOL self-supervised loss function tries to reduce the discrepancy in the two heads of the network when they are fed with differently augmented version of the same image. Although we use BYOL in this work, VINN can also work with other self-supervised representation learning methods [Chen et al. 2020c,d; Caron et al. 2020; Bardes et al. 2021] (Table 2.3).

In practice, we initialize both the BYOL online and target networks with an ImageNet-pretrained encoder. Then, using the BYOL objective, we finetune them to better fit our image distribution. Once the self-supervised training is done, we encode all our training demonstration frames with

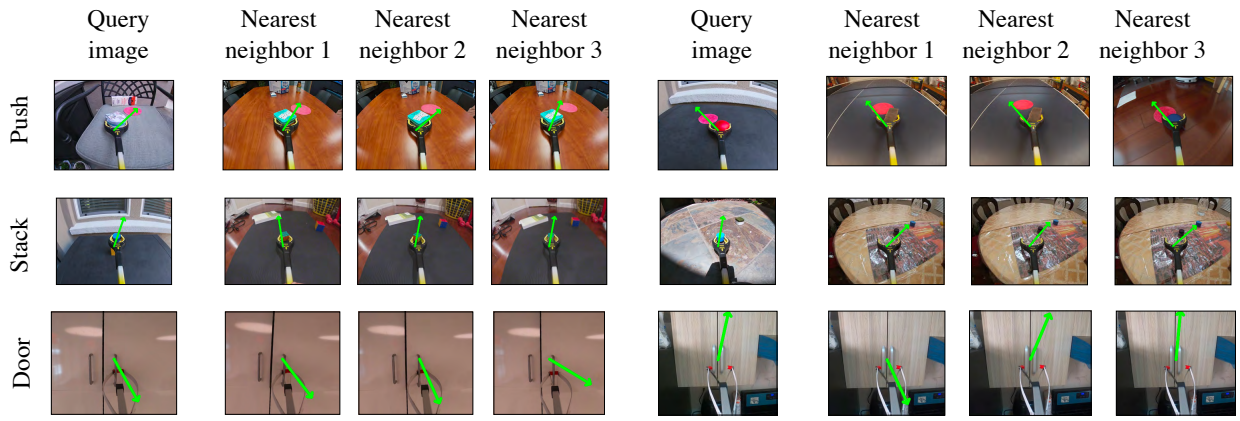


Figure 2.3: Nearest neighbor queries on the encoded demonstration dataset; the query image is on the first column, and the found nearest neighbors are on the next three columns. The associated action is shown with a green arrow. The bottom right set of nearest neighbors demonstrates the advantage of performing a weighted average over nearest neighbors' actions instead of copying the nearest neighbor's action.

the encoder to obtain a set of their embeddings, E.

2.3.2 k-Nearest Neighbors Based Locally Weighted Regression

The set of embeddings E given by our encoder holds compact representations of the demonstration images. Thus, during test time, given an input we search for demonstration frames with similar features. We find the nearest neighbors of the encoded input e from the set of demonstration embeddings, E. In Fig. 2.3, we see that these nearest neighbors are visually similar to the query image. Our algorithm implicitly assumes that a similar observation must result in a similar action. Thus, once we have found the k nearest neighbors of our query, we set the next action as an weighted average of the actions associated with those k nearest neighbors.

Concretely, this is done by performing nearest neighbors search based on the distance between embeddings: $\|e - e^{(i)}\|_2$, where $e^{(i)}$ is the i^{th} nearest neighbor. Once we find the k nearest neighbors and their associated actions, namely $(e^{(1)}, a^{(1)}), (e^{(2)}, a^{(2)}), \cdots, (e^{(k)}, a^{(k)})$, we set the action as the

Euclidean kernel weighted average [Atkeson et al. 1997] of those examples' associated actions:

$$\hat{a} = \frac{\sum_{i=1}^{k} \exp\left(-\|e - e^{(i)}\|_2\right) \cdot a^{(i)}}{\sum_{i=1}^{k} \exp\left(-\|e - e^{(i)}\|_2\right)}$$

In practice, this turns out to be the average of the observations' associated actions weighted by the SoftMin of their distance from the query image in the embedding space.

2.3.3 Deployment in real-robot door opening

For our robotic door opening task, we collect demonstrations using the DemoAT [Young et al. 2020] tool. Here, a reacher-grabber is mounted with a GoPro camera to collect a video of each trajectory. We pass the series of frames into a structure from motion (SfM) method which outputs the camera's location in a fixed frame [Ozyesil et al. 2017]. From the sequence of camera poses, which consist of coordinate and orientation, we extract translational motion which becomes our action. To extract the gripper state, we train a gripper network that outputs a distribution over four classes (open, almost open, almost closed, closed), which represent various stages of gripping. Then, we feed these images and their corresponding actions into our imitation learning method.

To train our visual encoders, we train ImageNet-pretrained BYOL encoders on individual frames in our demonstration dataset without action information. This same dataset with action information serves as the demonstration dataset for the k-NN based action prediction. Note that although we use task-specific demonstrations for representation learning, our framework is compatible with using other forms of unlabelled data such offline datasets [Gulcehre et al. 2020; Fu et al. 2020] or task-agnostic play data [Young et al. 2021].

To execute our door-opening skill on the robot, we run our model on a closed loop manner. After resetting the robot and the environment, on every step, we retrieve the robot observation and query the model with it. The model returns a translational action $\hat{a}$ as well as the gripper state

g, and the robot moves $c \odot \hat{a}$ where the vector c is a hyper-parameter with each element < 1 to mitigate our SfM model's inaccuracies and improve transfer from human demonstrations to robot execution. In addition, for nearest neighbor based methods, we have hyper-parameters that map the floating value g into a gripper state which was tuned per experiment.

2.4 Experimental Evaluation

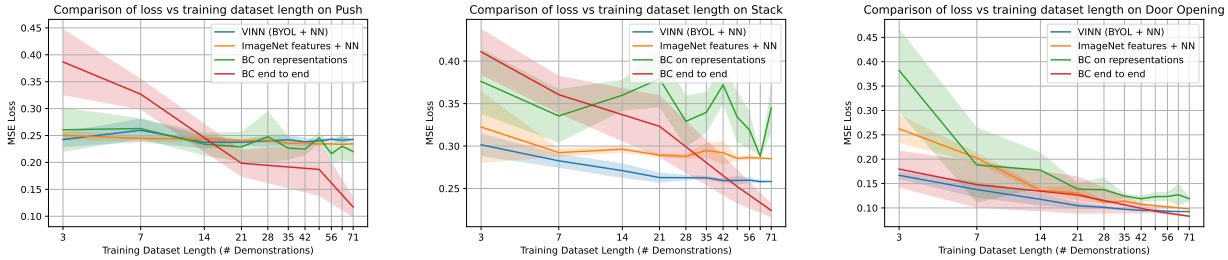


Figure 2.4: Mean Squared Error for the Pushing, Stacking and Door Opening (left to right) datasets of different algorithms trained on subsamples of the original dataset. End-to-end behavior cloning initialized with ImageNet-trained features perform as well as VINN for larger datasets, but fixed representation based methods outperforms it largely on small datasets.

In the previous sections we have described our framework for visual imitation, VINN. In this section, we seek to answer our key question: how well does VINN imitate human demonstrations? To answer this question, we will evaluate both on offline datasets and in closed-loop real-robot evaluation settings. Additionally, we will probe into the generalization with few demonstrations ability of VINN in settings where imitation algorithms usually suffer.

2.4.1 Experimental Setup

We conduct two different set of experiments: the first on the offline datasets for Pushing, Stacking and Door-Opening and the second on real-robot door opening.

Offline Visual Imitation Datasets Data for Pushing and Stacking tasks are taken from [Young et al. 2020]. The goal in the pushing task is to slide an object on a surface into a red circle. In the stacking task, the goal is to grasp an object present in the scene and move it on top of another object also in the scene, and release. To avoid confusion, in the expert demonstrations for stacking, the closest object is always placed on top of the distant object. The action labels are end-effector movements, which in this case is the translation vector in between the current frame and the subsequent one. In each case, there are a diverse set of backgrounds and objects that make up the scene and the task, making the tasks difficult.

For Door Opening, data is collected by 3 data-collectors in their kitchens. This amounts to a total of 71 demonstrations for training and 21 demonstrations for testing. We normalize all actions from the dataset to account for scale ambiguity from SfM. For all three tasks, we calculate the MSE loss between the ground truth actions and the actions predicted by each of the methods. Note that the number of demonstrations collected for this Door Opening task is an order of magnitude smaller than the ones used for Stacking and Pushing, which contain around 750 and 930 demonstrations respectively. To understand the performance on the various model in low data settings, we create subsampled Pushing and Stacking datasets containing 71 demonstrations on each for training and 21 for testing. This subsampling makes all three our datasets have the same size.

Closed-loop control We conduct our robot experiments on a loaded cabinet door opening task (see Fig. 2.1), where the goal of the robot is to grab hold of the cabinet handle and pull open the cabinet door. We use the Hello-Robot Stretch [Kemp et al. 2022] for this experiment. When evaluations start, the arm resets to ≈ 0.15 meters away from the cabinet door, with a random lateral translation within 0.05 meters parallel to the cabinet to evaluate generalization to varying starting states.

2.4.2 Baselines

We run our experiments for baseline comparison using the following methods:

- *Random Action:* In this baseline, we sample a random action from the action space.
- *Open Loop:* We find the maximum-likelihood open loop policy given all our demonstration, which is the average action $\overline{a}(t)$ over all actions $a_i(t)$ seen in the dataset at timestep t. In a Bayesian sense, if standard behavioral cloning is trying to approximate $p(a \mid s)$, this model is trying to approximate $p(a \mid t)$.
- *Behavioral Cloning (BC) end to end:* We train a ResNet-50 model with augmentated demonstration frames similar to [Torabi et al. 2018; Young et al. 2020]. We initialize the model with weights derived from ImageNet pretraining.
- *BC on Representations (BC-rep):* We use a self-supervised BYOL model to extract the encoding of each of our demonstration frames, and perform behavioral cloning on top of the representations. This baseline is similar to [Young et al. 2021] and performs better than end-to-end BC on the real robot (Table 2.1).
- *Implicit Behavioral Cloning:* We train Implicit BC [Florence et al. 2022] models on the tasks, modifying the official code.
- *ImageNet features + NN:* Instead of self-supervision, here we use the image representation generated by a pretrained ImageNet encoder akin to [Chen et al. 2020a]. The difference between this baseline and our method is simply forgoing the finetuning step on our dataset. This baseline highlights the importance of self-supervised pre-training on the domain related dataset.
- *Self-supervised learning method + NN:* This is our method; we compare three different ways of learning self-supervised representations features from our dataset – BYOL [Grill et al. 2020],

SimCLR [Chen et al. 2020c], and VICReg [Bardes et al. 2021], starting from an ImageNet pretrained ResNet-50, and then we use locally weighted regression to find the action.

2.4.3 Training Details

Each encoder network used in this paper follows the ResNet-50 architecture [He et al. 2016] with the final linear layer removed. Unless specified otherwise, we always initialize the weights of the ResNet-50 encoder with a pretrained model on ImageNet dataset. For VINN, we train our self-supervised encodings with the BYOL [Grill et al. 2020] loss. For standard end-to-end BC, we replace the last linear layer with a three-layer MLP and train it with the MSE loss. For BC-rep, we freeze the encoding network to the weights trained by BYOL on our dataset, and train just the final layers with the MSE loss. Additionally, for all visual learning, we use random crop, random color jitter, random grayscale augmentations and random blurring. We trained the self-supervised finetuning methods for 100 epochs on all three datasets.

2.4.4 How does VINN Perform on Offline Datasets?

For our first evaluation, we compare our method against the baselines on their Mean-Squared Error loss for the Pushing, Stacking, and Door-Opening tasks in Fig. 2.4. To understand the impact of the training dataset size on the algorithms, we run the models using multiple subsamples of different sizes from each dataset. We see that while end-to-end Behavioral Cloning starting from pretrained ImageNet representations can be better with a large amounts of training demonstrations, Nearest Neighbor methods are either competitive or better performing in low data settings.

On the Stacking and Door-Opening tasks, VINN is significantly better when the number of training demonstrations are small (< 20). While on the Pushing task, we notice that the task might be too difficult to solve with small number of demonstrations. One reason for this is that BYOL might

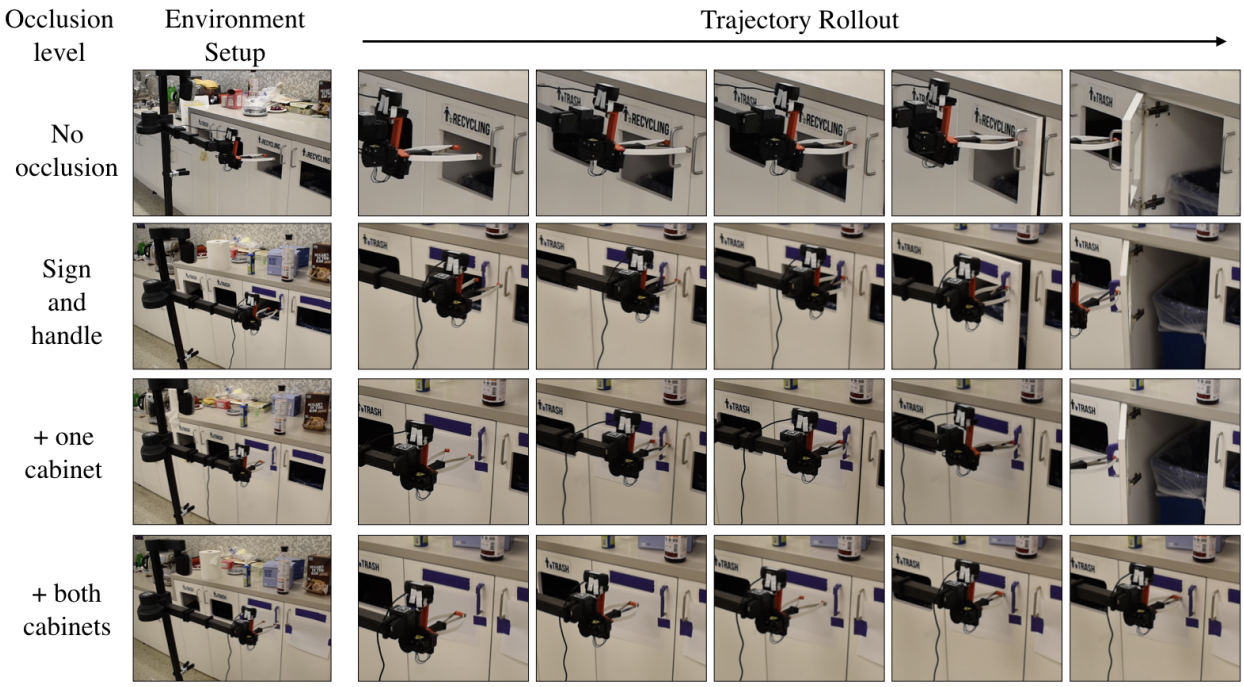


Figure 2.5: Sample frames from the rollouts from our model on the real robot experiments, with artificial occlusions added to the cabinet to test generalization. Under the maximum occlusion, our model fails to ever open the cabinet door, while in all other cases, the robot is able to succeed (Table 2.2.)

not be able to extract the most relevant representations for this task. Further experiments in Table 2.3 show that using other forms of self-supervision such as VICReg can significantly improve performance on this task. Overall, these experiments supports our hypothesis that provided with good representations, nearest-neighbor techniques can provide a competitive alternative to end-to-end behavior cloning.

2.4.5 How does VINN Perform on Robotic Evaluation?

Next, we run VINN and the baselines on our real robot environment. In this setting, our test environment comprises of the same three cabinets where training demonstrations were collected presented without any visual modifications. For each of our models, we run 30 rollouts with the robot in the real world with three different cabinets. On each rollout, the starting position of the

robot is randomized as detailed in (Sec. 2.4.1). In Table 2.1, we show the percentage of success from the 30 rollouts of each model, where we record both the number of time the robot successfully grasped the handle, as well as the number of time it fully opened the door.

Table 2.1: Success rate over 30 trials (10 trials on three cabinets each) on the robotic door opening task.

Method	Handle grasped	Door opened
BC (end to end)	0%	0%
BC on representations	56.7%	53.3%
Imagenet features + NN	20%	0%
VINN (BYOL + NN)	**80**%	**80**%

As we see from Table 2.1, VINN does better than all BC variants in successfully opening the cabinet door when there is minimal difference between the test and the train environments. Noticeably, it shows that depending on self-supervised features on augmented data make the models much more robust. BC, as an end-to-end parameteric model, does not have a strong prior on the actions if the robot makes a wrong move causing the visual observations to quickly goes out-of-distribution [Ross et al. 2011]. On the other hand, VINN can recover up to certain degree of deviation using the nearest neighbor prior, since the translation actions typically tend to re-center the robot instead of pushing it further out of distribution.

2.4.6 To What Extent does VINN Generalize to Novel Scenes?

To test generalization of our robot algorithms to novel scenes in the real world, we modified one of our test cabinets with various levels of occlusion. We show frames from a sample rollouts in each environment in Fig. 2.5, which also shows the cabinet modifications.

In Table 2.2, we see that VINN only completely fails when all the visual landscape on the cabinet is occluded. This failure is expected, because without coherent visual markers, the encoder fails to convey information, and thus the k-NN part also fails. Even then, we see that VINN succeeds at a higher rate even with significant modifications to the cabinet while BC-rep fails completely.

Table 2.2: Success rate over 10 trials on robotic door opening with visual modifications on one cabinet door.

Modification	BC-rep	VINN (ours)
Baseline (no modifications)	**90**%	80%
Covered signs and handle	10%	**70%**
Covered signs, handle, and one bin	0%	**50%**
Covered signs, handle, and both bins	0%	0%

Over all the real robot experiments, we find the following phenomenon: while a good MSE loss is not sufficient for a good performance in the real world, the two are still correlated, and a low MSE loss seems to be necessary for good real world performance. This observation let us test hypotheses offline before deploying and testing them in a real robot, which can be time-consuming and expensive. We hypothesize that this gap between performance on the MSE metric (Table 2.3) and real world performance (Table 2.1, 2.2) comes from variability in different models' ability to perform well in situations off the training manifold, where they may need to correct previous errors.

Table 2.3: Test MSE ($\times 10^{-1}$) on predicted actions for a set of baseline methods and ablations. Standard deviations, when reported, are over three randomly initialized runs.

			No Pretraining		With ImageNet Pretraining				
Tasks	Random	Open Loop	Implicit BC	BYOL + NN	BC-Rep	VINN (BYOL + NN)	VICREG + NN	SimCLR + NN	ImageNet + NN
Door Opening	6.34	2.27	1.8	1.52	1.19 ± 0.05	0.92	1.05	0.95	0.98
Stacking	6.13	2.83	7.1	2.82	3.45 ± 0.29	2.58	2.74	2.63	2.85
Pushing	6.15	2.12	5.6	2.43	2.20 ± 0.20	2.43	1.50	2.21	2.35

2.4.7 How Important are the Design Choices Made in VINN for Success?

VINN comprises of two primary components, the visual encoder and the nearest-neighbor based action modules. In this section, we consider some major design choices that we made for each of them.

Choosing the Right Self-supervision While we use a BYOL-based self-supervised encoding in our algorithm, there are multiple other self-supervised methods such as SimCLR and VICReg [Chen et al. 2020c; Bardes et al. 2021]. On a small set of experiments we noticed similar MSE losses compared to SimCLR [Chen et al. 2020c] and VICReg [Bardes et al. 2021]. From Table 2.3, we see that BYOL does the best in Door-Opening and Stacking, while VICReg does better in Pushing. However, we choose BYOL for our robot experiments since it requires less tuning overall.

Ablating Pretraining and Fine-tuning Another large gain in our algorithm is achieved by initializing our visual encoders with a network trained on ImageNet. In Table 2.3, we also show MSE losses from models that resulted from ablating this components of VINN. Removing this component achieves the column BYOL + NN (No Pretraining), which performs much worse than VINN. Similarly, the success of VINN depends on the self-supervised fine-tuning on our dataset, ablating which results in the model shown in ImageNet + NN column of Table 2.3. This model performs only slightly worse than VINN on the MSE metric. However, in Table 2.1, we see that this model performs poorly on the real world. These ablations show that the performance of our locally weighted regression based policy depends on the quality of the representation, where a good representation leads to better nearest neighbors, which in turn lead to a better policy both offline and online.

Performing Implicit instead of Explicit Imitation Moving away from the explicit forms of imitation where the models try to predict the actions directly, we run baselines with Implicit Behavioral Cloning (IBC) [Florence et al. 2022]. As we see on Table 2.3, this baseline fails to learn behaviors significantly better than the random or open loop baselines. We believe this is caused by two reasons. First, the implicit models have to model the energy for the full space (action space $\times$ observation space), which requires more data than the few demonstrations that we have in our datasets. Second, the official implementation of IBC supports $[-1, 1]^3$ as the action space instead

of its much smaller subspace of normalized 3d vectors S^2. This much larger action space, over which IBC tried to model the action, might have resulted in worse performance for IBC. While VINN makes the implicit assumption that the locally-weighted average of valid actions also yield a valid action, it can be freely projected to any relevant space without further processing, which makes it more flexible.

Learning a Parametric Policy on Representations Our Behavioral Cloning on representations (BC-Rep) baseline in all our experiments (Sec. 2.4) show the performance of a baseline where we use learned representations to learn a parametric behavioral policy. In the MSE losses (Table 2.3) and real world experiments (Table 2.1, 2.2.) This is the baseline that achieves the closest performance to VINN. However, the difference between BC-rep and VINN becomes more pronounced as the gap between training and test domain or the policy horizon grows. These experimental results indicate that using a non-parametric policy may be enabling us to be robust to out-of-distribution samples.

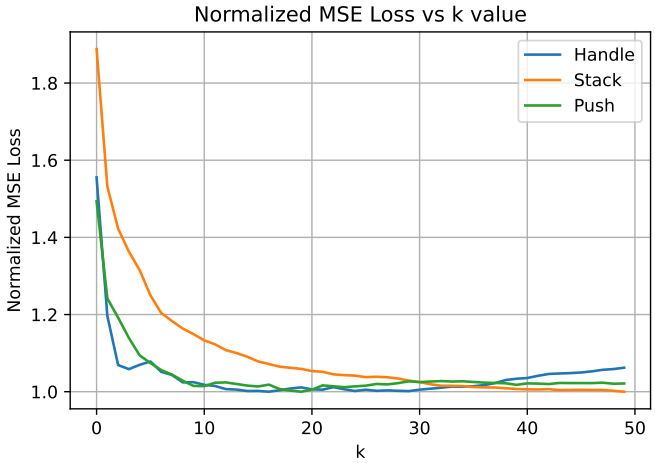


Figure 2.6: Value of k in the k-nearest neighbor weighted regression in VINN vs normalized MSE loss achieved by the model.

Choosing the Right k for k-Nearest Neighbors Finally, in VINN, we study the effect of different values of k for the k-NN based locally weighted controller. This parameter is important because with too small of a k, the predicted action may stop being smooth. On the other hand, with too large of a k, unrelated examples may start influencing the predicted action. By plotting our model's normalized MSE loss in the validation set against the value of k in Fig. 2.6, we find that around 10, k seems ideal for achieving low validation loss while averaging over only a few actions. Beyond $k = 20$, we didn't notice any significant improvement to our model from increasing k.

2.4.8 Computational Considerations

While the datasets we used for our experiments were not large, we recognize that our current nearest neighbor implementation is a $O(n)$ algorithm dependant linearly on the size of the training dataset with a naive algorithm. However, we believe VINN to be practical, since firstly, it was designed mostly for the small demonstration dataset regime where $O(n)$ is quite small, and secondly, this search can be sped up with a compiled index beyond the naive method using open-source libraries such as FAISS [Johnson et al. 2017] which were optimized to run nearest neighbor search on the order of billion examples [Matsui et al. 2018]. Currently, our algorithm takes ≈ 0.074 seconds to encode an image, and ≈ 0.038 seconds to perform nearest neighbors regression, which is only a small speed penalty for the robotic tasks we consider.

2.5 Limitations

In this work we proposed VINN, a new visual imitation framework that decouples visual representation learning from behavior learning. Although this decoupling improves over standard visual imitation methods, there are several avenues for future work. First, there is still some remaining hurdles to generalizing to a new scene, as seen in Sec. 2.4.6, where our model fails when all large,

recognizable markers are removed from the scene. While our NN-based action estimation lets us add new demonstrations easily, we cannot easily adapt our representation to such drastic changes in scene. An incremental representation learning algorithm has great potential to improve upon that. Second, our self-supervised learning is currently done on task related data, while ideally, if the dataset is expansive enough, task agnostic pre-training should also give us good performance [Young et al. 2021]. Finally, although our framework focuses on a single-task setting, we believe that learning a joint representation for multiple tasks could reduce the overall training overhead while being just as accurate.

Postscript

VINN is one of the most interesting works in this thesis, not because it provides a strong policy class lasting a century in the future, but because it shows us a fundamental truth about machine learning for policy learning – at some level, all algorithms do is nearest neighbor in some latent representation space. Building this mental model early in my Ph.D. helped me use behavior cloning judiciously to solve the problems it can solve. Simultaneously, I built a tool-belt of solutions I could reach for when BC would not be sufficient. This mental model has interesting implications for future work as well – for interpretability of our large models, for safety and data attribution works, and finally, for building interesting theoretical foundations for robot learning.

Acknowledgements

This work was co-led with Jyo Pari, co-authored with Sridhar Arunachalam Pandian, and advised by Lerrel Pinto. We thank Dhiraj Gandhi, Pete Florence, and Soumith Chintala for providing feedback on an early version of this paper. This work was supported by grants from Honda, Amazon, and ONR award numbers N00014-21-1-2404 and N00014-21-1-2758.

3 | Cloning k Behavior Modes with One Model: Behavior Transformers

3.1 Introduction

Creating agents that can behave intelligently in complex environments has been a longstanding problem in machine learning. Although Reinforcement Learning (RL) has made significant advances in behavior learning, its success comes at the cost of high sample complexity [Mnih et al. 2015; Duan et al. 2016; Akkaya et al. 2019]. Without priors on how to behave, state-of-the-art RL methods require online interactions on the order of 1-10M 'reward-labeled' samples for benchmark control tasks [Yarats et al. 2021a]. This is in stark contrast to vision and language tasks, where pretrained models and data-driven priors are the norm [Devlin et al. 2018; Brown et al. 2020; Grill et al. 2020; Bardes et al. 2021], which allows for efficient downstream task solving.

So how do we learn behavioral priors from pre-collected data? One option is offline RL [Levine et al. 2020], where offline datasets coupled with conservative policy optimization can learn task-specific behaviors. However, such methods have yet to tackle domains where task-specific reward labels are not present. Without explicit reward labels, imitation learning, particularly behavior cloning, is a more fitting option [Pomerleau 1989; Bojarski et al. 2016; Torabi et al. 2018]. Here, given behavior data $\mathcal{D} \equiv \{s_t, a_t\}$, behavior models can be trained to predict actions $f_\theta(s_t) \rightarrow a_t$

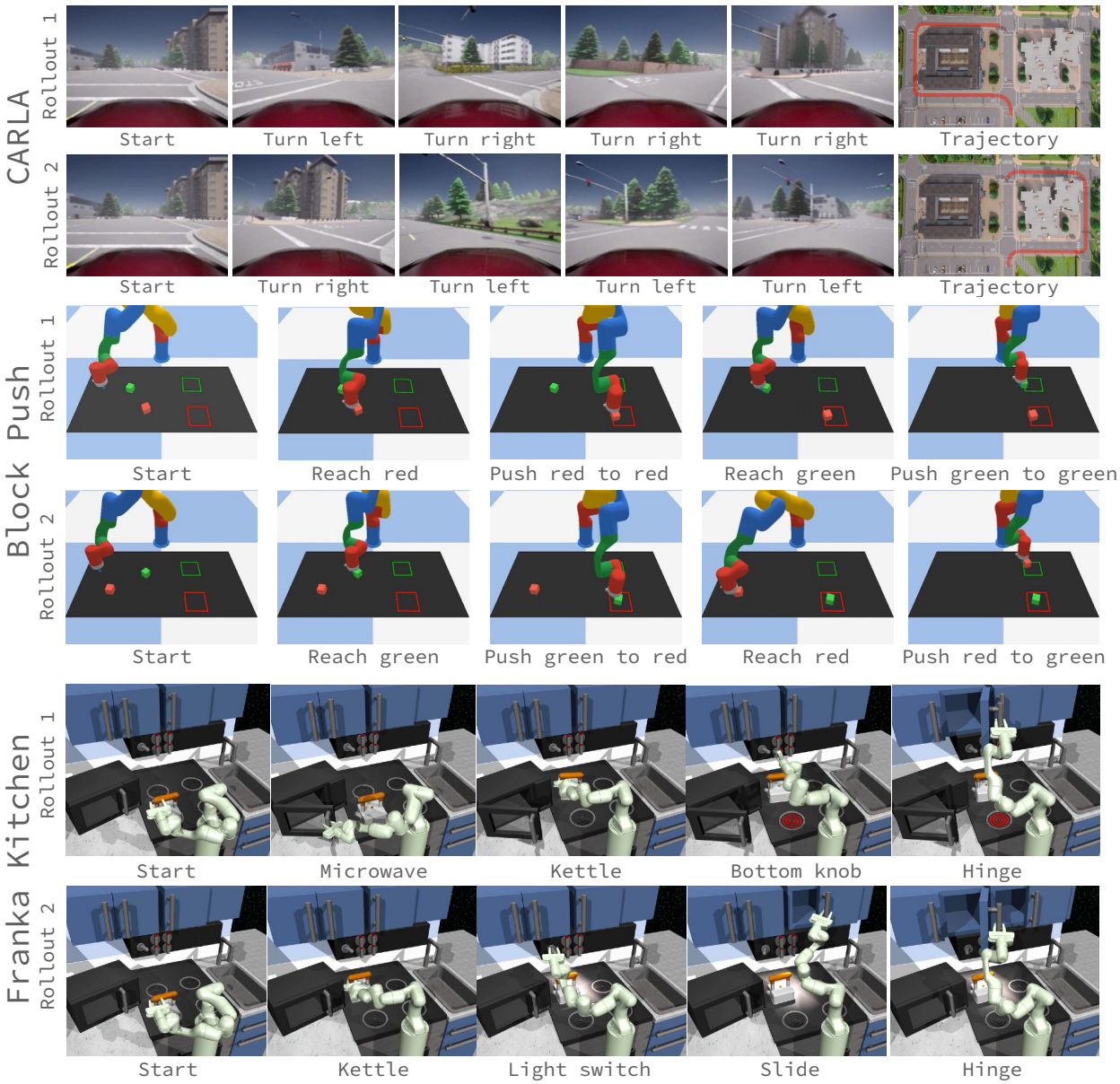


Figure 3.1: Unconditional rollouts from BeT models trained from multi-modal demonstartions on the CARLA, Block push, and Franka Kitchen environments. Due to the multi-modal architecture of BeT, even in the same environment successive rollouts can achieve different goals or the same goals in different ways.

through supervised learning. When demonstration data is plentiful, such approaches have found impressive success in a variety of domains from self-driving [Pomerleau 1989; Codevilla et al. 2019] to robotic manipulation [Zhang et al. 2018b; Pari et al. 2021]. Importantly, it requires neither online interactions nor reward labels.

However, state-of-the-art behavior cloning methods often make a fundamental assumption – that the data is drawn from a unimodal expert solving a single task. This assumption is often baked in to the architecture design, such as using a Gaussian prior. On the other hand, natural pre-collected data is sub-optimal, noisy, and contains multiple modes of behavior, all entangled in a single dataset. This distributionally multi-modal experience is most prominent in human demonstrations. Not only do we perform a large variety of behaviors every day, our personal biases result in significant multi-modality even for the same behavior [Grauman et al. 2021; Lynch et al. 2020]. Current approach for behavior cloning from such datasets primarily focus on learning goal-conditioned policies, where each goal implies a single mode of behavior [Hausman et al. 2017; Gupta et al. 2019; Lynch et al. 2020; Dasari and Gupta 2020]. However, even after goal-conditioning, an important question remains: How do we train models that can natively "clone" multi-modal behavior data?

In this work, we present Behavior Transformers (BeT), a new method for learning behaviors from rich, distributionally multi-modal data. BeT is based of three key insights. First, we leverage the context based multi-token prediction ability of transformer-based sequence models [Vaswani et al. 2017] to predict multi-modal actions. Second, since transformer-based sequence models are naturally suited to predicting discrete classes, we cluster continuous actions into discrete bins using k-means [MacQueen et al. 1967]. This allows us to model high-dimensional, continuous multi-modal action distributions as categorical distributions without learning complicated generative models [Kingma and Welling 2013; Dinh et al. 2016]. Third, to ensure that the actions sampled from BeT are useful for online rollouts, we concurrently learn a residual action corrector to produce

continuous actions for a sampled action bin.

We experimentally evaluate BeT on five datasets ranging from simple diagnostic toy datasets to complex datasets that include simulated robotic pushing [Florence et al. 2022], sequential task solving in kitchen environments [Gupta et al. 2019], and self-driving with visual observations in CARLA [Dosovitskiy et al. 2017]. The two main findings from these experiments can be summarized as:

1. On multi-modal datasets, BeT achieves significantly higher performance during online rollouts compared to prior behavior modelling methods.

2. Rather than collapsing or latching onto one mode, BeT is able to cover the major modes present in the training behavior datasets. Unconditional rollouts from this model can be seen in Fig. 3.1.

All of our datasets, code, and trained models will be made publicly available.

3.2 Behavior Transformers

Given a dataset of continuous observation and action pairs $\mathcal{D} \equiv \{(o, a)\} \subset \mathcal{O} \times \mathcal{A}$ that contains behaviors we are interested in, our goal is to learn a behavior policy $\pi : \mathcal{O} \mapsto \mathcal{A}$ that models this data without any online interactions with the environment or reward labels. This setup follows the Behavior Cloning formulation, where policies are trained to model demonstrations from expert rollouts. Often, such policies are chosen from a hypothesis class parametrized by parameter set θ. Following this convention, our objective is to find the parameter θ that maximizes the probability of the observed data

$$\theta^* := \arg\max_{\theta} \prod_t \mathbb{P}(a_t \mid o_t; \theta) \tag{3.1}$$

When the model class is restricted to unimodal isotropic Gaussians, this maximum likelihood estimation problem leads to minimizing the Mean Squared Error (MSE), $\sum_t \|a_t - \pi(o_t; \theta)\|^2$.

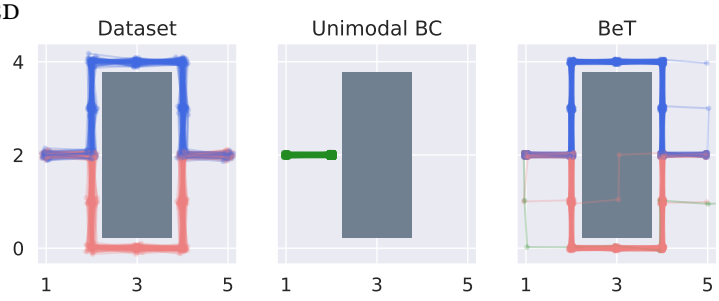


Figure 3.2: Comparison between a regular MSE-based BC model and a BeT models that can capture multi-modal distributions. The MSE-BC model takes 0 action to minimize MSE.

Limitations of traditional MSE-based BC: While MSE-based BC has been able to solve a variety of tasks [Bojarski et al. 2016; Torabi et al. 2018], it assumes that the data distribution is unimodal. Clean data from an expert demonstrator solving a particular task in a particular way satisfies this assumption, but pre-collected intelligent behavior often may not [Lynch et al. 2020; Gupta et al. 2019]. While more recent behavior generation models have sought to address this problem, they often require complex generative models [Singh et al. 2020], an exponential number of bins for actions [Mandi et al. 2021], complicated training schemes [Pertsch et al. 2021], or time-consuming test-time optimization [Florence et al. 2022]. An experimental analysis of some of these prior works is presented in Section 3.3.

Overview of Behavior Transformers (BeT): We address two critical assumptions in regular BC. First, we relax the assumption that the behavior we are cloning is purely Markovian, and instead model $P(a_t \mid o_t, o_{t-1}, \cdots, o_{t-h+1})$ for some horizon h. Second, instead of assuming that actions are generated by a unimodal action distribution, we model our action distribution as a mixture of gaussians. However, unlike previous efforts similar to Mixture Density Networks (MDN) to do so, whose limitations have been explored in Florence et al. [2022], we do not explicitly predict mode centers, which significantly improves our modeling capacity. To operationalize these two features in a single behavior model, we make use of transformers since (a) they are effective

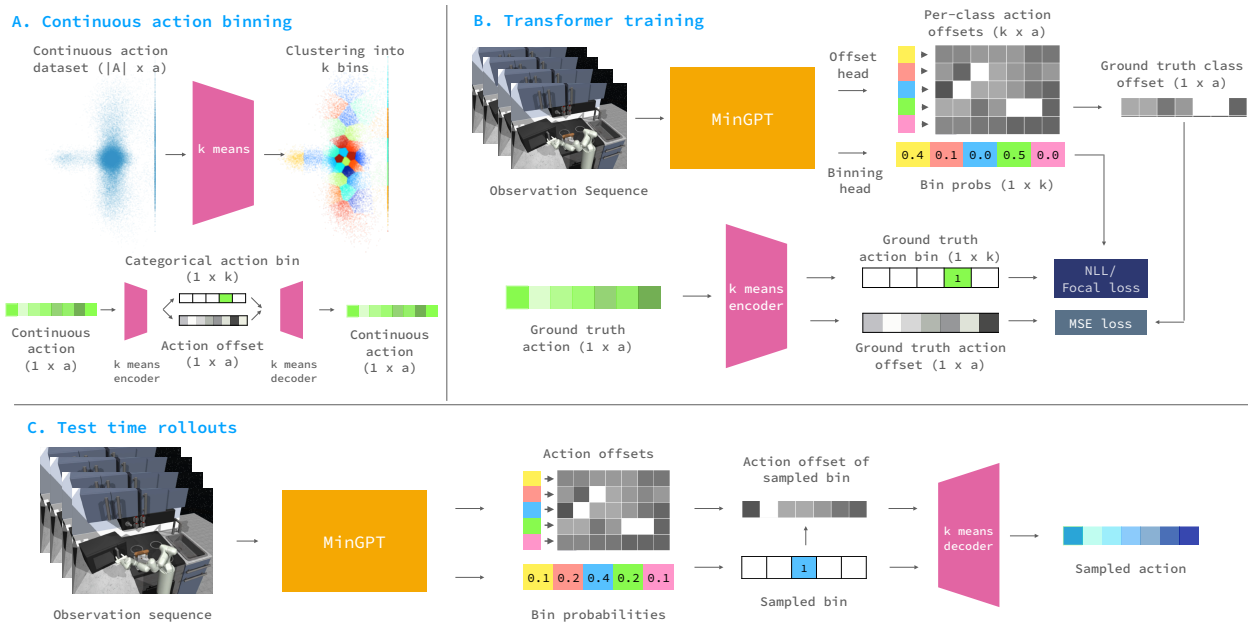


Figure 3.3: Architecture of Behavior Transformer. (A) The continuous action binning using k-means algorithm that lets BeT split every action into a discrete bin and a continuous offset, and later combine them into one full action. (B) Training BeT using demonstrations offline; each ground truth action provides a ground truth bin and residual action, which is used to train the minGPT trunk with its binning and action offset heads. (C) Rollouts from BeT in test time, where it first chooses a bin and then picks the corresponding offset to reconstruct a continuous action.

in utilizing prior observational history, and (b) they are naturally suited to output multi-modal tokens through their architecture.

3.2.1 Action discretization for distribution learning

Although transformers have become standard as a backbone for sequence-to-sequence models [Devlin et al. 2018; Brown et al. 2020], they are designed to process discrete tokens and not continuous values. In fact, modeling multi-modal distributions of high-dimensional continuous variables in a tractable manner is in itself a challenging problem, especially if we want the trained behavior model to cover the modes present in the dataset. To address this, we propose a new factoring of the action prediction task by dividing each action in two parts: a categorical variable denoting an 'action center', and a corresponding 'residual action'.

To this end, given the actions in our dataset, we first optimize for a set of k action centers, $\{A_1, A_2, \cdots, A_k\} \subset \mathcal{A}$. We then decompose each action into two parts: a categorical variable representing the closest action bin, $\lfloor a \rfloor := \arg\min_i \|a - A_i\|_2$, and a continuous residual action $\langle a \rangle := a - A_{\lfloor a \rfloor}$. If we are given the set of action centers $\{A_i\}_{i=1}^k$, an action bin index $\lfloor a \rfloor$ and the residual action $\langle a \rangle$, we can deterministically reconstruct the true action $a := A_{\lfloor a \rfloor} + \langle a \rangle$. Once learned, these k-means based encoder and decoders for this action factorization process are fixed for the rest of the train and testing phases. The action factorization procedure is illustrated in Fig. 3.3 (A).

3.2.2 Attention-based behavior mode learning

Once we have the clustering based autoencoder learned from the actions in the dataset, we model our demonstration trajectories with BeT. We use a transformer decoder model, namely minGPT [Brown et al. 2020], with minor modifications, as our backbone. The transformer $\mathcal{T}$ takes in a sequence of continuous observations $(o_i, o_{i+1}, \cdots, o_{i+h-1})$ and learns a sequence-to-sequence model mapping each observation to a categorical distribution over k discrete action bins. The predicted probability sequence is then compared with the ground truth labels, $(\lfloor a_i \rfloor, \lfloor a_{i+1} \rfloor, \lfloor a_{i+2} \rfloor, \cdots, \lfloor a_{i+h-1} \rfloor)$. We use a negative log-likelihood-based Focal loss [Lin et al. 2017] between the predicted categorical distribution probabilities and the ground truth labels to train the transformer head. Focal loss is a simple modification over the standard cross entropy loss. While the standard cross entropy loss for binary classification can be thought of $\mathcal{L}_{ce}(p_t) = -\log(p_t)$, Focal loss adds a term $(1 - p_t)^\gamma$ to this, to make the new loss

$$\mathcal{L}_{focal}(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

This loss has the interesting property that its gradient is more steep for smaller values of p_t, while flatter for larger values of p_t. Thus, it penalizes and changes the model more for making errors

in the low-probability classes, while is more lenient about making errors in the high probability classes. The model is illustrated in Fig. 3.3 (B).

3.2.3 Action correction: from coarse to finer-grained predictions

Using a transformer allows us to model multi-modal actions. However, discretizing the continuous action space in any way invariably causes loss of fidelity [Janner et al. 2021]. Discretization error may cause online rollouts of the behavior policy to go out of distribution from the original dataset [Ross et al. 2011], which can in turn cause critical failures. To predict the complete continuous action, we add an extra head to the transformer decoder that offsets the discretized action centers based on the observations.

For each observation o_i in the sequence, the head produces a $k \times \dim(A)$ matrix with k proposed residual action vectors, $\left(\langle a_i^{(j)} \rangle\right)_{j=1}^{k} = (\langle \hat{a}_i^{(1)} \rangle, \langle \hat{a}_i^{(2)} \rangle, \langle \hat{a}_i^{(3)} \rangle, \cdots, \langle \hat{a}_i^{(k)} \rangle)$, where the residual actions correspond to bin centers $A_1, A_2, A_3, \cdots, A_k$. These residual actions are trained with a loss akin to the *masked multi-task loss* [Girshick 2015] from object detection. In our case, if the ground truth action is $\mathbf{a}$, the loss is:

$$\text{MT-Loss}\left(\mathbf{a}, \left(\langle \hat{a}_i^{(j)} \rangle\right)_{j=1}^{k}\right) = \sum_{j=1}^{k} \mathbb{I}[\lfloor \mathbf{a} \rfloor = j] \cdot \|\langle \mathbf{a} \rangle - \langle \hat{a}^{(j)} \rangle\|_2^2 \tag{3.2}$$

Where $\mathbb{I}[]$ denotes the Iverson bracket, ensuring the offset head of BeT only incurs loss from the ground truth class of action $\mathbf{a}$. This mechanism prevents the model from trying to fit the ground truth action using the offset at every index.

3.2.4 Test-time sampling from BeT

During test time, at timestep t we input the latest h observations $(o_t, o_{t-1}, \cdots, o_{t-h+1})$ to the transformer, combining the present observation o_t with $h-1$ previous observations. Our trained

MinGPT model gives us $h \times 1 \times k$ bin center probability vectors, and $h \times k \times \dim(A)$ offset matrix. To sample an action at timestep t, we first sample an action center according to the predicted bin center probabilities on the t^{th} index. Once we have chosen an action center $A_{t,j}$, we add the corresponding residual action $\langle \hat{a}_t^{(j)} \rangle$ to it to recover a predicted continuous action $\hat{\mathbf{a}}_\mathbf{t} = A_{t,j} + \langle \hat{a}_t^{(j)} \rangle$. This sampling procedure is illustrated in Fig. 3.3 (C).

3.3 Experiments

We now study the empirical performance of BeT on a variety of behavior learning tasks. Our experiments are designed to answer the following questions: (a) Is BeT able to imitate multi-modal demonstrations? (b) How well does BeT capture the modes present in behavior data? (c) How important are the individual components of BeT?

3.3.1 Environments and datasets

We experiment with five broad environments. While full descriptions of these environments, dataset creation procedure, and overall statistics are in Appendix B.1, a brief description of them are as follows.

(a) **Point mass environment #1:** Our first set of experiments in Fig. 3.2, used to get a qualitative understanding of BeT, were performed in a simple Pointmass environment with a 2D observation and action space with two hundred demonstrations. The pre-collected demonstrations start at a fixed point, and then make their way to another point while avoiding a block in the middle. The two primary modes in this dataset are taking a left turn versus a right turn.

(b) **Point mass environment #2:** The setup is similar to the previous environment with the exception of one straight line and two complicated prolonged 'Z' shaped modes of demonstration

(Fig. 3.5.)

(c) **CARLA self-driving environment:** CARLA [Dosovitskiy et al. 2017] uses the Unreal Engine to provide a simulated driving environment in a visually realistic landscape. The agent action space is 2D (accelerate/brake and left/right steer), while the observation space is (224,224,3)-dimensional RGB image from the car. A hundred total demonstrations drive around a building block in two distinct modes. This environment highlights the challenge of behavior learning from high-dimensional observations as shown in Fig. 3.1 (a). For visual observations with BeT, we use a frozen ResNet-18 [He et al. 2016] pretrained on ImageNet [Deng et al. 2009] as an encoder.

(d) **Multi-modal block-pushing environment:** For more complicated interaction data, we use the multi-modal block-pushing environment from Implicit Behavioral Cloning (IBC) [Florence et al. 2022], where an XArm robot needs to push two blocks into two squares in any order. The blocks and target squares are colored red and green. The positions of the blocks are randomized at episode start. We collect 1,000 demonstrations using a deterministic controller with two independent axes of multi-modality: (a) it starts by reaching for either the red or the green block, with 50% probability, and (b) it pushes the blocks to (red, green) or (green, red) squares respectively with 50% probability.

(e) **Franka kitchen environment:** To highlight the complexity of performing long sequences of actions, we use the Relay Kitchen Environment [Gupta et al. 2019] where a Franka robot manipulates a virtual kitchen environment. We use the relay policy learning dataset with 566 demonstrations collected by human participants wearing VR headsets. The participants completed a sequence of four object-interaction tasks in each episode [Gupta et al. 2019]. There are a total of seven interactable objects in the kitchen: a microwave, a kettle, a slide cabinet, a hinge cabinet, a light switch, and two burner knobs. This dataset contains two different kinds of multi-modality: one from the inherent noise in human demonstrations, and

another from the demonstrators' intent.

3.3.2 Baseline behavior learning methods

While a full description of our baselines are in Appendix B.2.1, a brief description of them is here:

(a) **Multi-layer Perceptron with MSE (RBC):** We use MLP networks trained with MSE loss as our first baseline, since this is the standard way of performing behavioral cloning for a new task [Torabi et al. 2018]. A comparison with transformer-based behavior cloning is discussed in Section 3.3.5.

(b) **Nearest neighbor (NN):** Nearest neighbor based algorithms are easy to implement, and has recently shown to have strong performance on complicated behavioral cloning tasks [Arunachalam et al. 2023b].

(c) **Locally Weighted Regression (LWR):** This non-parametric approach provides better regularization compared to NN and is a strong alternative to parametric BC [Atkeson et al. 1997; Pari et al. 2021].

(d) **Variational auto-encoders (VAE):** Inspired by SPiRL [Pertsch et al. 2021], where behavioral priors are learned through a VAE [Kingma and Welling 2013], we compare with continuous actions generated from the VAE and the prior.

(e) **Normalizing Flow (Flow):** Inspired by PARROT [Singh et al. 2020], where state-conditioned action priors are learned through a Flow model [Dinh et al. 2016], we compare with actions generated from the Flow model.

(f) **Implicit Behavioral Cloning (IBC):** Instead of modeling the conditional distribution $P(a \mid o)$, IBC models the joint probability distribution $P(a, o)$ using energy-based models [Florence et al. 2022]. While IBC is slower than explicit BC models because of their sampling

Table 3.1: Performance of BeT compared with different baselines in learning from demonstrations. For CARLA, we measure the probability of the car reaching the goal successfully. For Block push, we measure the probability of reaching one and two blocks, and the probabilities of pushing one and two blocks to respective squares. For Kitchen, we measure the probability of n tasks being completed by the model within the allotted 280 timesteps. Evaluations are over 100 rollouts in CARLA and 1,000 rollouts in Block push and Kitchen environments.

	CARLA	Block push				Kitchen				
	Driving	Reach		Push		# Tasks completed				
Baselines	Success	R1	R2	P1	P2	1	2	3	4	5
RBC	0.98	0.67	0	0	0	0	0	0	0	0
1-NN	0.99	0.49	0.05	0.01	0	0.90	0.72	0.44	0.17	0
LWR	**1**	0.50	0.06	0	0	**1**	0.83	0.52	0.21	0
VAE	0	0.60	0.05	0	0	**1**	0	0	0	0
Flow	0.03	0.59	0.02	0	0	0.04	0	0	0	0
IBC	0.25	0.98	0.04	0.01	0	0.99	0.87	0.61	0.24	0
BeT (Ours)	0.98	**1**	**0.99**	**0.96**	**0.71**	0.99	**0.93**	**0.71**	**0.44**	**0.02**

requirements, they have been shown to learn well on multi-modal data, and outperform earlier work such as MLP-MDNs [Bishop 1994].

3.3.3 Is BeT able to imitate multi-modal demonstrations?

The first question we ask is whether BeT can actually clone behaviors given a mixed dataset of unlabeled, multi-modal behaviors. To examine that, we look at the performance of our model in CARLA, Block push, and Kitchen environments compared with our baselines in Table 3.1.

We see that BeT outperforms all other methods in all environments except CARLA, where it is narrowly outperformed by LWR. Since the models are all behavioral cloning algorithms, they share the failure mode of failing once the observations go out of distribution (OOD). However, they vary in the tolerance. For example, BeT shines in the Block push environment, where alongside extreme environment randomness and multi-modality, the models also have to learn significant long-term behaviors and commit to a single mode over a long period. While all baselines can somewhat successfully reach one block, they fail to complete the long-horizon, multi-modal task

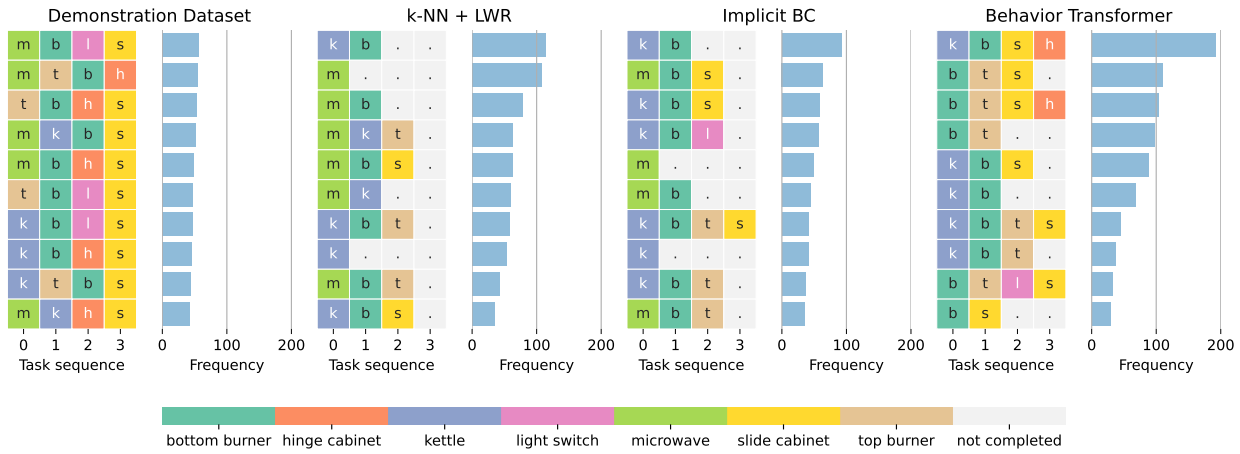


Figure 3.4: Distribution of most frequent tasks completed in sequence in the Kitchen environment. Each task is colored differently, and frequency is shown out of a 1,000 unconditional rollouts from the models.

of pushing two blocks into two different bins. On the other hand, we observe that BeT's primary failure mode is not realizing a block has not completely entered the target yet, while other methods either go OOD quickly, or keep switching between modes. We also observe that BeT performs well even in complex observation and action spaces. In the CARLA environment, the model takes in visual observations, while in the Franka Kitchen environment, the action space corresponds to a 9-DOF torque controlled robot. BeT handles both cases with the same ease as it does environments with lower-dimensional observation or action spaces.

3.3.4 Does BeT capture the modes present in behavior data?

Next, we examine the question of whether, given a dataset where multi-modal behavior exists, our model learns behavior that is also multi-modal. Here, we are interested in seeing the variance of the behavior of the model over different rollouts. In each of our environments, the demonstrations contain different types of multi-modality. As a result, we show a comprehensive analysis of multi-modality seen in our agent behaviors.

We see in Table 3.2 that in CARLA and Block push, BeT covers all the modes of the demonstration

Table 3.2: Multimodality learned from the multimodal demonstrations by different algorithms. In CARLA, we consider the probability of turning left vs. right at the intersection, ignoring OOD rollouts. In Block push, we consider two set of probabilities, (a) which block was reached first, and (b) what was the pushing target for each block. Finally, in Franka Kitchen, we consider the empirical entropy for the task sequences, considered as strings, sampled from the model. We highlight the values closest to the corresponding demonstration values.

	CARLA		Block: first block reached		Push: red block target		Push: green block target		Kitchen
Baselines	Left	Right	Red	Green	Red	Green	Red	Green	Task entropy
RBC	0	0.98	0.41	0.25	0	0	0	0	0
1-NN	0	0.99	0.24	0.25	0	0	0	0.01	2.12
LWR	0	1	0.26	0.26	0.01	0	0.01	0.01	2.29
VAE	0	0	0.27	0.33	0	0	0	0	0.72
Flow	0	0	0.31	0.29	0	0	0	0	0.08
IBC	0.12	0.13	**0.48**	**0.50**	0	0	0.01	0.01	2.41
BeT (Ours)	**0.34**	**0.64**	0.54	0.46	**0.43**	**0.44**	**0.41**	**0.40**	**2.47**
Demonstrations	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	2.96

data, even in the few cases where it does not perfectly match the demonstrated task probabilities. For the Kitchen environment, we see in Fig. 3.4 that BeT visits certain strings of tasks more frequently than in the original demonstrations. However, compared to other strong baselines, BeT generates longer task strings more often while maintaining diversity and not collapsing to a single mode.

3.3.5 How important are the individual components of BeT?

There are four key differences between BeT architecture and standard BC: (a) binning actions into discrete clusters, (b) using offsets to faithfully reconstruct actions later, (c) learning sequentially to use historical context, and (d) using an attention-based MinGPT trunk. In this section, we discuss the impacts they have in BeT performance.

Impact of discrete binning: Intuitively, having discrete options for bin centers is what enables BeT to express multi-modal behavior even when starting from an identical starting state. Indeed,

Table 3.3: Relative performance of ablated variants of BeT, normalized by average BeT successes at the task

Ablations	CARLA	Block push	Kitchen
No offsets	0.94	0.95	0.78
No binning	0.94	0.25	0.68
No history	0.65	0.95	0.88
MLP	0.90	0	0.05
Temp. Conv	0.72	0.01	0.26
LSTM	0.03	0.03	0.04
GPT-MDN	0.30	0.83	0.86
Unif. quant.	0.90	0.96	0.90

if there is no binning, we see from Table 3.3 that the performance of BeT drops significantly. More tellingly, in the Franka Kitchen environment, the model only ever completed a subsequence of (kettle, top/bottom burner, light switch, slide cabinet) tasks after 100 random rollouts. This result shows us that having discrete bins helps BeT achieve multi-modality. We also experiment with the Mixture density networks (MDN) [Bishop 1994] and uniform quantization, as shown in previous works [Florence et al. 2022; Janner et al. 2021]. We see that they may perform well sometimes but overall still fall short of our k-means binning approach.

Necessity of action offsets: An important feature of BeT is the residual action offset that corrects the discrete actions coming from the bins. While the bin centers may be quite expressive, Table 3.3 shows that the inability to correct them causes a performance degrade. Interestingly, the largest degradation comes in the Kitchen environment, which also has the highest dimensional action space. Intuitively, we can understand how in higher dimension the loss of fidelity from discretizing would be higher, and the relative performance loss across three environments support that hypothesis.

Importance of historical context: While RL algorithms traditionally assume environments are Markovian, human behavior in an open-ended environment is rarely so. Thus, using historical

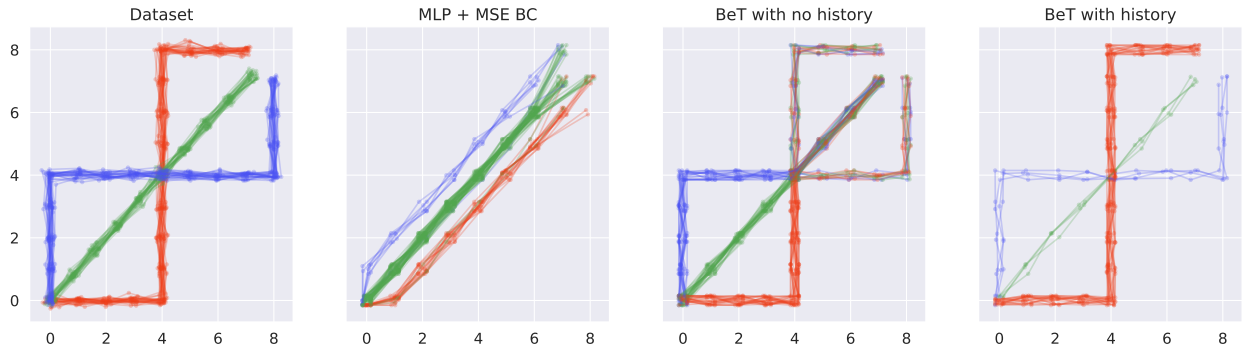


Figure 3.5: Comparison between an RBC model and two BeT models, trained with and without historical context on a dataset with three distinct modes. BeT with history is better able to capture the context-dependant behavior in the demonstrations.

context helps BeT to perform well. We show a simple experiment in Fig. 3.5 on the second point mass environment. Here, training and evaluating with some historical context allows BeT to follow the demonstrations better. We experience the same in the CARLA, Block push, and Kitchen environments, where training with some historical context raises performance across the board as seen in Table 3.3.

Importance of transformer architecture: Despite transformers' success in other fields of machine learning, it is natural to wonder whether the tasks BeT solves here really requires one. We ablated BeT by replacing the MinGPT trunk with an MLP, Temporal Convolution, and LSTMs, and found that they have lower performance while also being difficult to train stably. This performance reduction remains even if the MLP is given some historical context by stacking h observations before passing it to the MLP. See Table. 3.3 for results and Appendix B.3.2 for further details.

Ablating the number of discrete bin centers, k: Since BeT is trained with a sum of focal loss for the binning head and MSE loss for the offset head, the number of cluster centers present a trade-off in the architecture. Concretely, as the number of bins go up, the log-likelihood loss goes up but the MSE loss goes down. In Table 3.3, we showed that using only one bin ($k = 1$) decreases

the performance level of BeT.

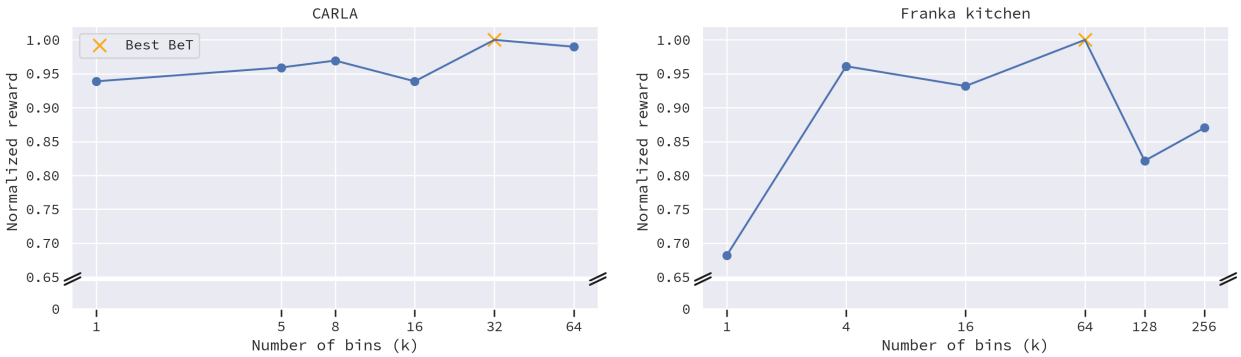


Figure 3.6: Ablating the number of discrete bin centers k for BeT. Reward is normalized with respect to the best performing model.

In Fig. 3.6, we present the plot of the variation in performance as k value changes. We see that for BeT it matters somewhat to pick a number of clusters k that is in the right neighborhood. However, the range for near-optimum performance is quite wide. In our experiments, we also pick a k in the right neighborhood and only run a sweep at the very end to find out an optimal value for k.

Computation considerations: While transformers in usual contexts are large models, we downscale them for our application in BeT (See Appendix B.2.4). Our models contain on the order of 10^4–10^6 parameters, and even with a small batch size trains within an hour for our largest datasets (Block push) on a single desktop GPU. In contrast, for the same task, our strongest baseline IBC takes about 14 hours. Evaluation rollouts on the same environment take 1.65 seconds with BeT, as opposed to 17.70 seconds with IBC.

3.4 Related Work

This paper builds upon a rich literature in imitation learning, offline learning, generative models, and transformer architectures. The most relevant ones to our work are discussed here.

Learning from offline data: Since Pomerleau [1988] showed the possibility of driving an autonomous vehicle using offline data and a neural network, learning behavior from offline data has been a continuous topic of research for scalable behavior learning [Argall et al. 2009; Billard et al. 2008; Schaal 1999]. The approaches can be divided into two broad classes: Offline RL [Fujimoto et al. 2018; Kumar et al. 2019, 2020; Wu et al. 2019; Levine et al. 2020; Fu et al. 2020], focusing on learning from datasets of a mixed quality that also have reward labels; and imitation learning [Osa et al. 2018; Peng et al. 2018, 2021; Ho and Ermon 2016], focusing on learning behavior from a dataset of expert behavior without reward labels. BeT falls under the second category, as it is a behavior cloning model. Behavior cloning is a form of imitation learning that tries to model the action of the expert given the observation which is often used in real-world applications [Zhang et al. 2018b; Zhu et al. 2018; Zhang et al. 2018b; Rahmatizadeh et al. 2018; Florence et al. 2019; Zeng et al. 2020]. As behavior cloning algorithms are generally solving a fully supervised learning problem, they tend to be faster and simpler than reinforcement learning or offline RL algorithms and in some cases show competitive results [Fu et al. 2020; Gulcehre et al. 2020].

Generative models for behavior learning: One approach for imitation learning is Inverse Reinforcement Learning or IRL [Russell 1998; Ng et al. 2000], where given expert demonstrations, a model tries to construct the reward function. This reward function is then used to generate desirable behavior. GAIL [Ho and Ermon 2016], an IRL algorithm, connects generative adversarial models with imitation learning to construct a model that can generate expert-like behavior. Under this IRL framework, previous works have tried to predict multi-modal, multi-human trajectories [Lee and Kitani 2016; Ivanovic et al. 2018]. Similarly, other works have tried Gaussian Processes [Rasmussen and Nickisch 2010] for creating dynamical models for human motion [Wang et al. 2007]. Another class of algorithms learn a generative action decoder [Pertsch et al. 2021; Lynch et al. 2020; Singh et al. 2020] from interaction data to make downstream reinforcement learning faster and easier, which inspired BeT's action factorization. Finally, a class of algorithms, most

notably [Liu et al. 2020; Florence et al. 2022; Kostrikov et al. 2021; Nachum and Yang 2021] do not directly learn a generative model but instead learn energy based models. These energy based models can then be sampled to generate desired behavior. Since [Florence et al. 2022] is a BC model capable of multi-modality, we compare against it as a baseline in Sec. 3.3.

Transformers for control: With the stellar success of transformer models [Vaswani et al. 2017] in natural language processing [Devlin et al. 2018; Brown et al. 2020] and computer vision [Dosovitskiy et al. 2020], there has been significant interest in using transformer models to learn behavior and control. Among those, [Chen et al. 2021; Janner et al. 2021] applies them to Reinforcement Learning and Offline Reinforcement Learning, respectively, while [Clever et al. 2021; Dasari and Gupta 2020; Mandi et al. 2021] use them for imitation learning. [Dasari and Gupta 2020; Mandi et al. 2021] use transformers mostly to summarize historical visual context, while [Clever et al. 2021] relies on their long-term extrapolation abilities to collect human-in-the-loop demonstrations more efficiently. BeT is inspired by both of these use cases, as we use a transformer to summarize historical context while leveraging its generative abilities. Architecturally, BeT is most closely related to the imitation learning variant of [Janner et al. 2021], with a significant difference that while [Janner et al. 2021] learns the joint state, action distribution, BeT learns the conditional distribution of action given state, which allows BeT to tackle much more complicated state spaces.

Datasets for distributionally multi-modal data: Similar to computer vision [Deng et al. 2009; Lin et al. 2014; Liu et al. 2018] and natural language processing [Bowman et al. 2015; Rajpurkar et al. 2016], there has been a recent interest in collecting behavior datasets that may aid in downstream behavior learning. Some of them are labeled with agent goals or rewards for downstream tasks [Mandlekar et al. 2018; Fu et al. 2020; Mandlekar et al. 2021], while others are more open ended [Gupta et al. 2019; Lynch et al. 2020; Young et al. 2021] and come without

reward or task labels. In our work, we focus towards the latter class. The lack of labeled goal or reward labels in the second category implies that there is more multi-modality in the action distributions compared to action distributions of goal or reward conditioned datasets, which is the same reason a lot of work learning from multi-modal datasets try to learn a goal-conditioned model [Hausman et al. 2017; Gupta et al. 2019; Lynch et al. 2020; Dasari and Gupta 2020]. Finally, the lack of labelling requirements mean that the unlabelled datasets are cheaper to obtain, which should help BeT scale further in the future.

3.5 Limitations

In this work, we introduce Behavior Transformers (BeT), which uses a transformer-decoder based backbone with a discrete action mode predictor coupled with a continuous action offset corrector to model continuous actions sequences from open-ended, multi-modal demonstrations. While BeT shows promise, the truly exciting use of it would be to learn diverse behavior from human demonstrations or interactions in the real world. In parallel, extracting a particular, unimodal behavior policy from BeT during online interactions, either by distilling the model or by generating the right 'prompts' [Reynolds and McDonell 2021], would make BeT tremendously useful as a prior for online Reinforcement Learning.

Postscript

With hindsight worthy of a full thesis, the most important contribution of BeT in literature is pushing the line of mainstream research into multi-modal behavior cloning further. The earliest work focusing primarily on this problem was Implicit Behavior Cloning (IBC), and BeT creates a system that is much more usable in practice without the optimization issues present in earlier work. Even today, a lot of the benchmarks (Franka Kitchen, BlockPush) and metrics (behavior

entropy) used to evaluate multi-modal policy learning originated in this work.

What held BeT back at that time was a lack of strong baselines. At that time, we were elated to outperform the strongest baseline (IBC or GMM) by a lot, and thus did not know that a much stronger algorithm is possible with small modifications to the method Chapter 5. Relative to this, Diffusion Policy had a much stronger baseline to work off of which helped it optimize the algorithm to its best version. Another limitation, that is yet to be addressed, is the interaction of the hybrid action representation with reinforcement learning algorithms. After a number of years using purely BC training on such policy architecture, the natural next step is RL self-improvement, but the hybrid action space makes it more complex than using only continuous or discrete action spaces.

4 | Conditional Behavior Generation from Uncurated Robot Data: Conditional Behavior Transformers

4.1 Introduction

Machine Learning is undergoing a Cambrian explosion in large generative models for applications across vision [Ramesh et al. 2022] and language [Brown et al. 2020]. A shared property across these models is that they are trained on large and uncurated data, often scraped from the internet. Interestingly, although these models are trained without explicit task-specific labels in a self-supervised manner, they demonstrate a preternatural ability to generalize by simply conditioning the model on desirable outputs (e.g. "prompts" in text or image generation). Yet, the success of conditional generation from uncurated data has remained elusive for decision making problems, particularly in robotic behavior generation.

To address this gap in behavior generation, several works [Lynch et al. 2020; Pertsch et al. 2021] have studied the use of generative models on *play* data. Here, play data is a form of offline, uncurated data that comes from either humans or a set of expert policies interacting with the environment. However, once trained, many of these generative models require significant amounts

Table 4.1: Comparison between existing algorithms to learn from large, uncurated datasets: GCBC [Lynch et al. 2020], GCSL [Ghosh et al. 2019], Offline GCRL [Ma et al. 2022b], Decision Transformer [Chen et al. 2021]

	GCBC	GCSL	Offline RL	Decision Transformer	C-BeT (ours)
Reward-free	✓	✓	✗	✗	✓
Offline	✓	✗	✓	✓	✓
Multi-modal	✗	✗	✗	✗	✓

of additional online training with task-specific rewards [Gupta et al. 2019; Singh et al. 2020]. In order to obtain task-specific policies without online training, a new line of approaches employ offline RL to learn goal-conditioned policies [Levine et al. 2020; Ma et al. 2022b]. These methods often require rewards or reward functions to accompany the data, either specified during data collection or inferred through hand-crafted distance metrics, for compatibility with RL training. Unfortunately, for many real-world applications, data does not readily come with rewards. This prompts the question: *how do we learn conditional models for behavior generation from reward-free, play data?*

To answer this question, we turn towards transformer-based generative models that are commonplace in text generation. Here, given a prompt, models like GPT-3 [Brown et al. 2020] can generate text that coherently follow or satisfy the prompt. However, directly applying such models to behavior generation requires overcoming two significant challenges. First, unlike the discrete tokens used in text generation, behavior generation will need models that can output continuous actions while also modeling any multi-modality present in the underlying data. Second, unlike textual prompts that serve as conditioning for text generation, behavior generation may not have the condition and the operand be part of the same token set, and may instead require conditioning on future outcomes.

In this work, we present Conditional Behavior Transformers (C-BeT), a new model for learning conditional behaviors from offline data. To produce a distribution over continuous actions instead of discrete tokens, C-BeT augments standard text generation transformers with the action

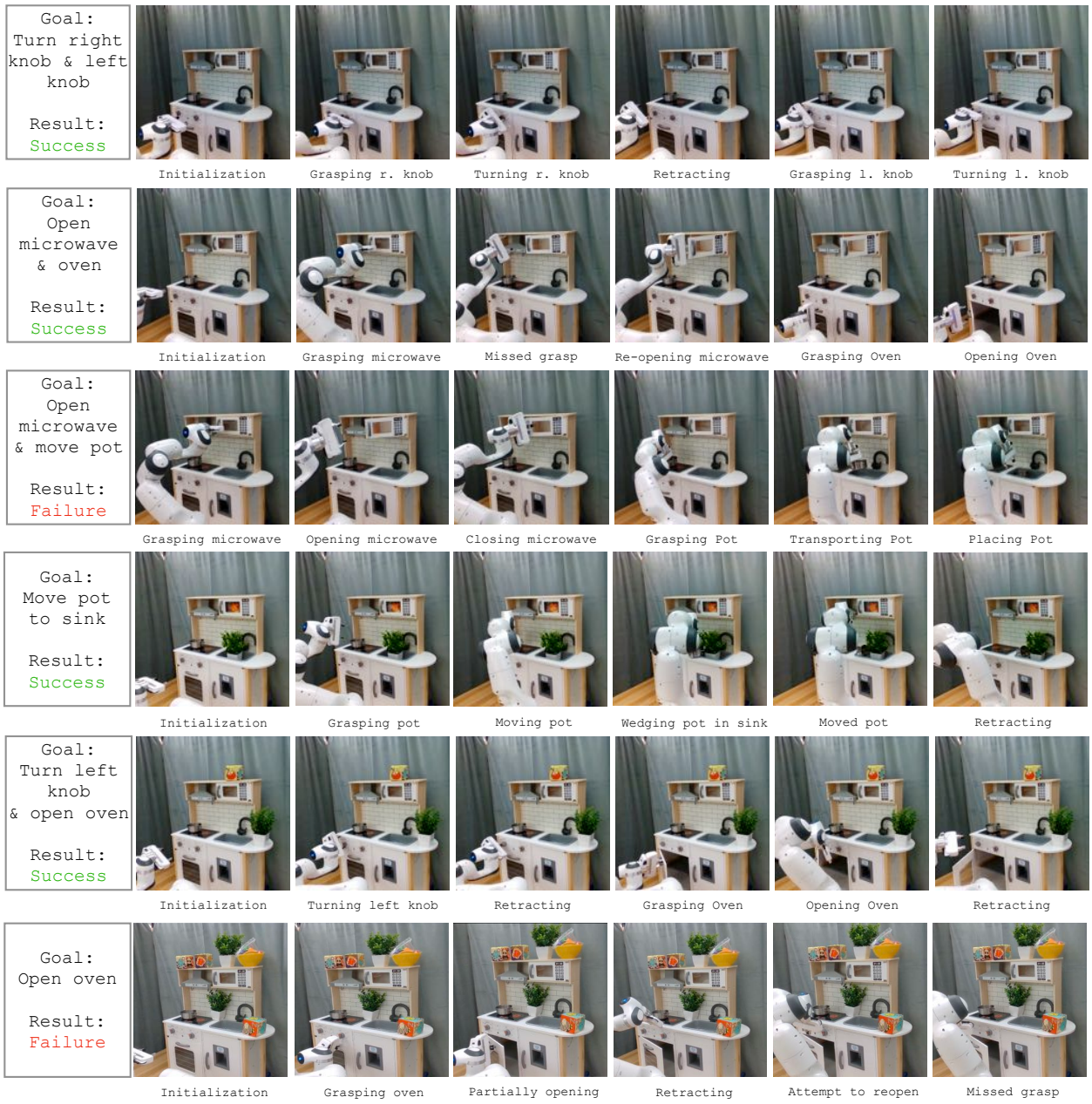


Figure 4.1: Multiple conditioned roll-outs of visual robot policies learned on our toy kitchen with only 4.5 hours of human play interactions. Our model learns purely from image and proprioception without human labeling or data curation. During evaluation, the policy can be conditioned either on a goal observation or a demonstration. Note that the last three rows contain distractor objects in the environment that were never seen during training.

discretization introduced in Behavior Transformers (BeT) [Shafiullah et al. 2022]. Conditioning in C-BeT is done by specifying desired future states as input similar to Play-Goal Conditioned Behavior Cloning (Play-GCBC) [Lynch et al. 2020]. By combining these two ideas, C-BeT is able to leverage the multi-modal generation capabilities of transformer models with the future conditioning capabilities of conditional policy learning. Importantly, C-BeT does not require any online environment interactions during training, nor the specification of rewards or Q functions needed in offline RL.

We experimentally evaluate C-BeT on three simulated benchmarks (visual self-driving in CARLA [Dosovitskiy et al. 2017], multi-modal block pushing [Florence et al. 2022], and simulated kitchen [Gupta et al. 2019]), and on a real Franka robot trained with play data collected by human volunteers. The main findings from these experiments can be summarized as:

1. On future-conditioned tasks, C-BeT achieves significantly higher performance compared to prior work in learning from play.

2. C-BeT demonstrates that competent visual policies for real-world tasks can be learned from fully offline multi-modal play data (rollouts visualized in Figure 4.1).

4.2 Background and Preliminaries

Play-like data: Learning from Demonstrations [Argall et al. 2009] is one of the earliest frameworks explored for behavior learning algorithms from offline data. Typically, the datasets used in these frameworks have a built in assumption that the demonstrations are collected from an expert repeatedly demonstrating a single task in exactly the same way. On the contrary, play datasets violate many of such assumptions, like those of expertise of the demonstrator, and the unimodality of the task and the demonstrations. Algorithms that learn from such datasets sometimes assume that the demonstrations collected are from a rational agent with possibly some latent intent in

their behavior [Lynch et al. 2020]. Note that, unlike standard offline-RL datasets [Fu et al. 2020], play-like behavior datasets neither contain fully random behaviors, nor have rewards associated with the demonstrations.

Behavior Transformers (BeT): BeT [Shafiullah et al. 2022] is a multi-modal behavior cloning model designed particularly for tackling play-like behavior datasets. BeT uses a GPT-like transformer architecture to model the probability distribution of action given a sequence of states $\pi(a_t \mid s_{t-h:t})$ from a given dataset. However, unlike previous behavior learning algorithms, BeT does not assume a unimodal prior for the action distribution. Instead, it uses a k-means discretization to bin the actions from the demonstration set into k bins, and then uses the bins to decompose each action into a discrete and continuous component. This support for multi-modal action distributions make BeT particularly suited for multi-modal, play-like behavior datasets where unimodal behavior cloning algorithms fail. However, vanilla BeT only supports unconditonal behavior rollouts, which means that it is not possible to choose a targeted mode of behavior during BeT policy execution.

Conditional behavior learning: Generally, the problem of behavior learning for an agent is considered the task of learning a *policy* $\pi : \mathcal{O} \rightarrow \mathcal{A}$ mapping from the environment observations to the agent's actions that elicit some desired behavior. Conditional behavior learning is concerned with learning a policy $\pi : \mathcal{O} \times \mathcal{G} \rightarrow \mathcal{A}$ conditioned additionally on a secondary variable g sampled from a distribution $p(g)$. This condition variable could be specific environment states, latents (such as one-hot vectors), or even image observations. The success of a conditioned policy can be evaluated either through pre-specified reward functions, distance function between achieved outcome g' and specified outcome g, or by discounted visitation probability $d_{\pi(\cdot|g)} = \mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t \delta(\phi(o_t) = g)]$ if a mapping ϕ between states and achieved outcome is defined [Eysenbach et al. 2022].

Goal Conditioned Behavior Cloning (GCBC): In GCBC [Lynch et al. 2020; Emmons et al. 2021], the agent is presented with a dataset of (observation, action, goal) tuples (o, a, g), or sequences

of such tuples, and the objective of the agent is to learn a goal-conditioned behavior policy. The simplest way to achieve so is by training a policy $\pi(\cdot \mid o, g)$ that maximizes the probability of the seen data $\pi^* = \arg\max_\pi \prod_{(o,a,g)} \mathbb{P}[a \sim \pi(\cdot \mid o, g)]$. Assuming a unimodal Gaussian distribution for $\pi(a \mid o, g)$ and a model parametrized by θ, this comes down to finding the parameter θ minimizing the MSE loss, $\theta^* = \arg\min_\theta \sum_{(o,a,g)} ||a - \pi(o, g; \theta)||^2$. To make GCBC compatible with play data that inherently does not have goal labels, goal relabeling from future states is often necessary. A common form of data augmentation in training such models, useful when $\mathcal{G} \subset \mathcal{O}$, is hindsight data relabeling [Andrychowicz et al. 2017], where the dataset $\{(o, a, g)\}$ is augmented with $\{(o_t, a, o_{t'}) \mid t' > t\}$ by relabeling any reached state in a future timestep as a goal state and adding it to the dataset.

4.3 Approach

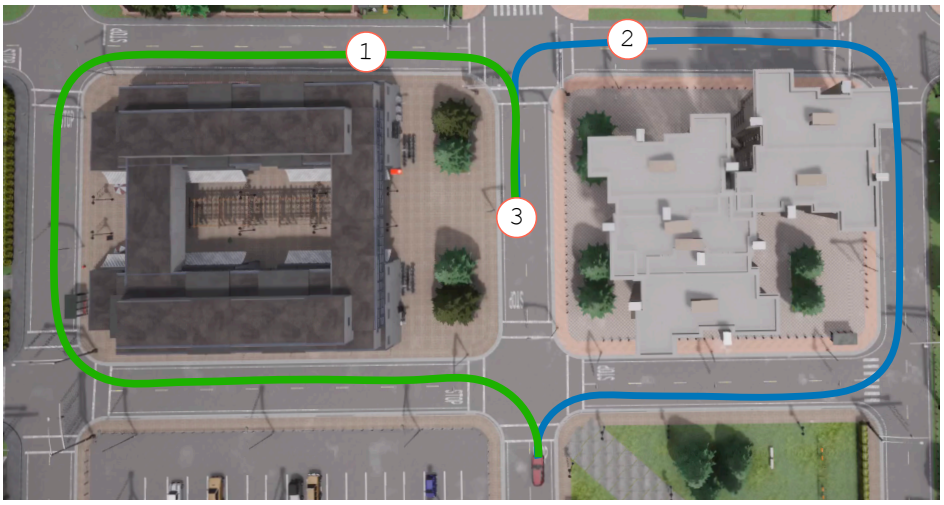


Figure 4.2: Conditional behavior learning from play demonstrations. Here, a policy conditioned on reaching ① or ② has only one possible course of action, but conditioned on reaching ③ there are two reasonable paths.

Given a dataset $\{(o, a)\} \in \mathcal{O} \times \mathcal{A}$ of sequences of (observation, action) pairs from a play dataset, our goal is to learn a behavior generation model that is capable of handling multiple tasks and multiple

ways of accomplishing each task. At the same time, we wish to be able to extract desired behavior from the dataset in the form of a policy through our model, or, in terms of generative models, "controllably generate" our desired behavior (see Figure 4.2). Finally, in the process of learning this controllable, conditional generative model, we wish to minimize the amount of additional human annotation or curation required in preparing the dataset. The method we develop to address these needs is called Conditional Behavior Transformer.

4.3.1 Conditional Behavior Transformers (C-BeT)

Conditional task formulation: First, we formulate the task of learning from a play dataset as learning a conditional behavior policy, i.e. given the current state, we need to model the distribution of actions that can lead to particular future states. For simplicity, our formulation can be expressed as $\pi : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{D}(\mathcal{A})$ where, given a current observation o_c and a future observation o_g, our policy π models the distribution of the possible actions that can take the agent from o_c to o_g. Mathematically, given a set of play trajectories T, we model the distribution $\pi(a \mid o_c, o_g) \triangleq \mathbb{P}_{\tau \in T}(a \mid o_c = \tau_t, o_g = \tau_{t'}, t' > t)$. Next, to make our policy more robust since we operate in the partially observable setting, we replace singular observations with a sequence of observations; namely replacing o_c and o_g with $\bar{o}_c = o_c^{(1:N)}$ and $\bar{o}_g = o_g^{(1:N)}$ for some integer N. Thus, the final task formulation becomes learning a generative model π with:

$$\pi\left(a \mid o_c^{(1:N)}, o_g^{(1:N)}\right) \triangleq \mathbb{P}_{\tau \in T}\left(a \mid o_c^{(1:N)} = \tau_{t:t+N}, o_g^{(1:N)} = \tau_{t':t'+N}, t' > t\right) \tag{4.1}$$

Architecture selection: Note that the model for our task described in the previous paragraph is necessarily multi-modal, since depending on the sequences $\bar{o}_c$ and $\bar{o}_g$, there could be multiple plausible sequences of actions with non-zero probability mass. As a result, we choose Behavior Transformers (BeT) [Shafiullah et al. 2022] as our generative architecture base as it can learn action generation with multiple modes. We modify the input to the BeT to be a concatenation

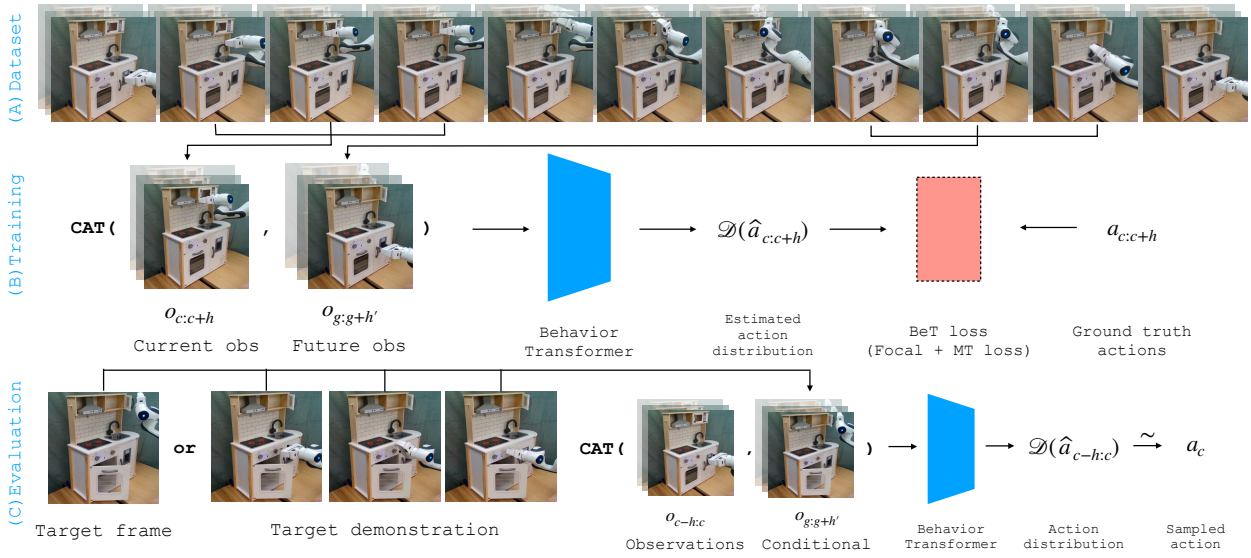


Figure 4.3: End-to-end training and evaluation of C-BeT. (A) Our dataset consists of play data in an environment, which may contain semi-optimal behavior, multi-modal demonstrations, and failures, and does not contain any annotations or task labels. (B) We train our C-BeT model by conditioning on current and future states using BeT (Section 4.2) (C) During evaluation, our algorithm can be conditioned by target observations or newly collected demonstrations to generate targeted behavior.

of our future conditional observation sequence and current observation sequence. We choose to concatenate the inputs instead of stacking them, as this allows us to independently choose sequence lengths for the current and future conditional observations. Since BeT is a sequence-to-sequence model, we only consider the actions associated with the current observations as our actions. We show the detailed architecture of our model in Figure 4.3.

Dataset preparation: To train a C-BeT model on our play dataset $\{(o, a)\}$, we will need to appropriately prepare the dataset. We first convert the dataset to hold sequences of observations associated with actions, $\{(o_{t:t+N}, a_{t:t+N})\}$. Then, during training time, we dynamically augment each pair with a sequence of future observations, functionally converting our dataset into $\{(o_{t:t+N}, a_{t:t+N}, o_{t':t'+N'})\}$ for some $t' > t$, and treat the sequence $o_{t':t'+N'}$ as $\bar{o}_g$.

Training objective: We employ the same objective as BeT in training C-BeT. For each of the current observation and future conditional pair, we compute the BeT loss (see appendix C.1 for details) between the ground truth actions and the predicted actions. We compute the focal loss [Lin

et al. 2017] on the predicted action bins, and the MT-loss [Girshick 2015] on the predicted action offsets corresponding to the action bins as described in BeT.

Test-time conditioning with C-BeT: During test time, we again concatenate our future conditional sequence with our current observations, and sample actions from our model according to the BeT framework. While in this work, we primarily condition C-BeT on future observations, we also study other ways of training and conditioning it, such as binary latent vectors denoting the modes in a trajectory in our experiments, and compare its performance to observation-conditioned C-BeT (see Section 4.4.5).

4.4 C-BeT on Simulated Benchmarks

In this section, we discuss our experiments in simulation that are designed to answer the following key questions: How well does C-BeT learn behaviors from play? How important is multi-modal action modeling? And finally, how does C-BeT compare to other forms of conditioning?

4.4.1 Baselines

We compare with the following state-of-the-art methods in learning from reward-free offline data:

- **Goal Conditioned BC (GCBC):** GCBC [Lynch et al. 2020; Emmons et al. 2021] learns a policy by optimizing the probability of seen actions given current and the end state in a trajectory.
- **Weighted Goal Conditioned Supervised Learning (WGCSL) [Yang et al. 2022]:** GCSL [Ghosh et al. 2019] is an online algorithm with multiple rounds of collecting online data, relabeling, and training a policy on that data using GCBC. WGCSL [Yang et al. 2022] improves GCSL by learning an additional value function used to weight the GCSL loss. We compare against an single-round, offline variant of WGCSL in this work.

- **Learning Motor Primitives from Play (Play-LMP):** Play-LMP [Lynch et al. 2020] is a behavior generation algorithm that focuses on learning short ($\sim$ 30 timesteps) motor primitives from play data. Play-LMP does so by using a variational-autoencoder (VAE) to encode action sequences into motor program latents and decoding actions from them.

- **Relay Imitation Learning (RIL):** Relay Imitation Learning [Gupta et al. 2019] is a hierarchical imitation learning with a high level controller that generates short term target state given long term goals, and a low level controller that generates action given short term target.

- **Conditional Implicit Behavioral Cloning (C-IBC):** Implicit behavioral cloning [Florence et al. 2022] learns an energy based model (EBM) $E(a \mid o)$ over demos and during test samples action a given an observation o. We compare against a conditional IBC by training an EBM $E(a \mid o, g)$.

- **Generalization Through Imitation (GTI):** GTI [Mandlekar et al. 2020] encodes the goal condition using a CVAE, and autoregressively rolls out action sequences given observation and goal-latent. We follow their architecture and forgo collecting new trajectories with an intermediate model since that does not fit an offline framework.

- **Offline Goal-Conditioned RL:** While offline RL is generally incompatible with play data without rewards, recently some offline goal-conditioned RL algorithms achieved success by optimizing for a proxy reward defined through state occupancy. Our baseline, GoFAR [Ma et al. 2022b], is one such algorithm that learns a goal-conditioned value function and optimizes a policy to maximize it.

- **Behavior Transformers (BeT):** We include unconditional BeT (Sec. 4.2) in our baseline to understand the improvements made by the C-BeT conditioning. In practice, it acts as a "random" baseline that performs the tasks without regard for the goal.

Table 4.2: Results of future-conditioned algorithms on a set of simulated environments. The numbers reported for CARLA, BlockPush, and Kitchen are out of 1, 1, and 4 respectively, following [Shafiullah et al. 2022]. In CARLA, success counts as reaching the location corresponding to the observation; for BlockPush, it is pushing one or both blocks into the target squares; and for Kitchen, success corresponds to the number of conditioned tasks, out of four, completed successfully.

	GCBC	WGCSL	Play-LMP	RIL	C-IBC	GTI	GoFAR	BeT	C-BeT (unimodal)	C-BeT (multimodal)
CARLA	0.04	0.02	0.0	0.59	0.65	0.74	0.72	0.31	0.62	**0.98**
BlockPush	0.06	0.10	0.02	0.07	0.01	0.04	0.04	0.34	0.35	**0.90**
Kitchen	0.74	1.17	0.04	0.39	0.13	1.61	1.24	1.77	2.74	**2.80**

- **Unimodal C-BeT:** We use our method without the multi-modal head introduced in BeT. This also corresponds to a variant of Decision Transformer conditioning on outcomes instead of rewards.

Note that neither WGCSL nor GoFAR are directly compatible with image states and goals, since they require a proxy reward function $r : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$. Thus, we had to design a proxy reward function on the image representations, $\exp\left(-(1/4||g - s||)^2\right)$ to apply them on image-based environments. For a fair comparison, we also upgrade baseline Play-LMP, C-IBC, and GTI architectures by giving them sequences of observations and retrofitting them with transformers whenever applicable.

4.4.2 Simulated Environments and Datasets

We run our algorithms and baselines on a collection of simulated environments as a benchmark to select the best algorithms to run on our real robotic setup. The simulated environments are selected to cover a variety of properties that are necessary for the real world environment, such as pixel-based observations, diverse modes in the play dataset, and complex action spaces (see Figure. 4.4).

1. **CARLA self-driving:** CARLA [Dosovitskiy et al. 2017] is a simulated self-driving environment created using Unreal Engine. In this environment, the observations are RGB pixel values of

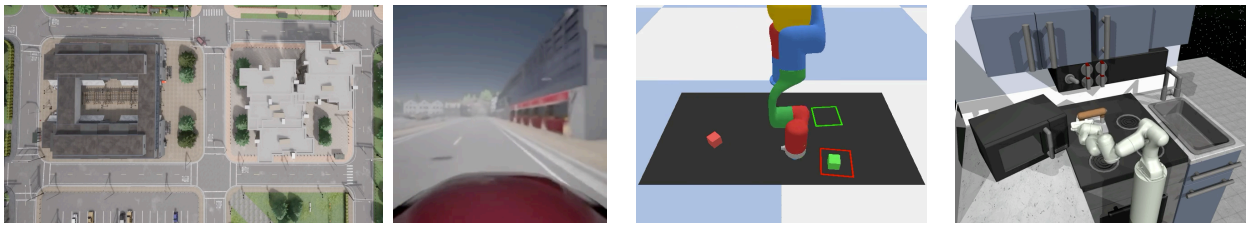


Figure 4.4: Visualizations of simulated environments that we evaluate our methods on, from left to right: CARLA self-driving (top down view and agent POV), BlockPush, and Franka Kitchen.

dimension $(224, 224, 3)$, and actions are two-dimensional (accelerate/brake and steer). We use an environment with a fork in the road (see Figure 4.2) following two possible routes to the same goal, collecting 200 demonstrations in total. We condition on one of the two possible routes to the goal, and at the goal where choosing either of the two modes is valid.

2. **Multi-modal block-pushing:** We use the multi-modal block-pushing environment from [Florence et al. 2022] for complicated multi-modal demonstrations. In this environment, an xArm robot pushes two blocks, red and green, into two square targets colored red and green. All positions are randomized with some noise at episode start. We use 1,000 demonstrations collected using a deterministic controller, and condition on just the future block positions on each baseline.

3. **Franka relay kitchen:** Originally introduced in [Gupta et al. 2019], Relay Kitchen is a robotic environment in a simulated kitchen with seven possible tasks. A Franka Panda robot is used to manipulate the kitchen, and the associated dataset comes with 566 demonstrations collected by humans with VR controllers performing four of the seven tasks in some sequence.

4.4.3 How well does C-BeT learn behaviors from play?

On each of these environments, we train conditional behavior generation models and evaluate them on a set of conditions sampled from the dataset. The success is defined by the model

performing the same tasks as conditioned by the future outcome. We see from Table. 4.2 that C-BeT performs significantly better compared to the baselines on all three tasks. BeT, as our unconditioned "random" baseline, shows the success rate of completing tasks unconditionally, and see that none of the baselines surpasses it consistently. Out of the MLP-based baselines, WGCSL performs best in the state-based tasks. However, GoFAR performs best on the CARLA vision based environment where the other two MLP-based baselines fail almost completely. We note that Play-LMP performs poorly because our tasks are long-horizon and quite far from its intended motor primitive regime, which may be challenging for Play-LMP's short-horizon auto-encoding architecture.

4.4.4 How important is multi-modal action modeling?

While we use a multi-modal behavior model in this work, it is not immediately obvious that it may be necessary. Specifically, some previous outcome-conditioned policy learning works [Chen et al. 2021; Emmons et al. 2021] implicitly assume that policies are unimodal once conditioned on an outcome. In Table 4.2 the comparison between C-BeT and unimodal C-BeT shows that this assumption may not be true for all environments, and all else being equal, having an explicitly multi-modal model helps learning an outcome conditioned policy when there may be multiple ways to achieve an outcome.

4.4.5 How does C-BeT compare to other forms of conditioning?

Table 4.3: Comparison between C-BeT with no supervised labels and labels acquired with human supervision.

	No labels	Labels
CARLA	0.98	1.0
BlockPush	0.90	0.89
Kitchen	2.80	2.75

We consider the question of how much comparative advantage there is in getting human labels for our tasks. We do so by adding manual one-hot (CARLA, BlockPush) or binary (Kitchen) labels to our tasks, and training and evaluating C-BeT with those labels. As we see on Table 4.3, on the three simulated environments, C-BeT conditioned on only future observations performs comparably to conditioning with human labels.

4.5 C-BeT on Real-World Robotic Manipulation

We now discuss our robot experiments, which are geared towards understanding the usefulness of C-BeT on real-world play data.

4.5.1 Robotic Environment and Dataset

Robot setup: Our environment consists of a Franka Emika Panda robot, similar to the simulated Franka Kitchen environment, set up with a children's toy kitchen set (see Figure 4.1). The toy kitchen has an oven, a microwave, a pot, and two stove knobs that are relevant to our play dataset. The action space in this environment contains the seven joint angle deltas normalized within the $[-1, 1]$ range, and a binary gripper control.

Play dataset: We collected 460 sequences totaling to 265 minutes (about 4.5 hours) of play data on the toy kitchen with volunteers using a Vive VR controller to move the Franka. While collecting the play data, we did not give the volunteers any explicit instructions about doing any particular tasks, or number of tasks, beyond specifying the interactable items, and stipulating that the pot only goes on the left stove or in the sink, to prevent dropping the pot and reaching an unresettable state. As the observations, we save the RGB observations from two cameras on the left and right of the setup, as well as the robot's proprioceptive joint angles. Overall, the dataset contains 45 287

Table 4.4: Single-task success rate in a real world kitchen with conditional models. We present the success rate and number of trials on each task, with cumulative results presented on the last column.

	Knobs	Oven	Microwave	Pot	Cumulative
GoFAR	0/10	0/5	0/5	0/5	0/25
Unconditional BeT	5/20	6/10	1/10	0/10	12/50
Unimodal C-BeT	1/20	8/10	4/10	0/10	13/50
Multimodal C-BeT	3/20	9/10	7/10	5/10	24/50

frames of play interactions and their associated actions.

Representation learning To simplify the task of learning policies on image space, we decouple the task of image representation learning from policy learning following [Pari et al. 2021]. For each camera, we first fine-tune a pretrained ResNet-18 [He et al. 2016] encoder on the acquired frames with BYOL self-supervision [Grill et al. 2020]. Then, during policy learning and evaluation, instead of the image from the cameras, we pass the two 512-dimensional BYOL embeddings as part of the observation. For the proprioceptive part of the observation, we repeat the $(\sin, \cos)$ of seven joint states 74 times to get a 1036-dimensional proprioceptive representation, making our overall observation representation 2060-dimensional.

4.5.2 Conditional Behavior Generation on Real Robot

Behavior generation on single tasks: Our first experiment in the real robot is about extracting single-task policies from the play dataset. We define our tasks as manipulating the four types of interactable objects one at a time: opening the oven door, opening the microwave door, moving the pot from the stove to the sink, and rotating a knob 90 degrees to the right. We use appropriate conditioning frames from our observation dataset, and start the robot from the neutral state to complete the four tasks. The result of this experiment is presented in Table 4.4. We see that on single task conditionals, C-BeT is able to complete all tasks except the knobs consistently, outperforming all our baselines, showing that C-BeT is able to extract single-task policies out of

Table 4.5: Task success rate in a real world kitchen with conditional models evaluated on a long-horizon goal. We present the success rate and number of trials on each task, with cumulative result presented on the last column.

	Oven → Pot	Microwave → Oven	Pot → Microwave	Avg. Tasks/Run
Unconditional BeT	$(6, 0)/10$	$(1, 6)/10$	$(0, 1)/10$	0.47
Unimodal C-BeT	$(1, 1)/10$	$(2, 0)/10$	$(8, 0)/10$	0.37
Multimodal C-BeT	$(5, 4)/10$	$(8, 8)/10$	$(4, 4)/10$	1.1

uncurated, real-world play data. We discuss failures of C-BeT on the knob tasks in Section 4.5.3. While our GoFAR baseline was able to move towards the task targets, it was unable to successfully grasp or interact with any of the target objects. We believe it may be the case because unlike the robot experiment in [Ma et al. 2022b], we do not have the underlying environment state, the tasks are much more complicated, and our dataset is an order of magnitude smaller (400 K vs 45 K).

Behavior generation for longer horizons: Next, we ask how well our models work for longer-horizon conditioning with multiple tasks. We choose play sequences from the dataset with multiple tasks completed and use their associated states as the conditions for our models. In our roll-outs, we calculate how many tasks completed in the original sequence were also completed in the conditional roll-outs. We calculate this metric over 3 conditioning sequences, and report the results in Table 4.5. We see that even without any high level controller, C-BeT is able to stitch together multiple tasks from play demonstrations to complete long-horizon goals.

Generalization to prompt and environment perturbations: A major requirement from any robot system deployed in the real world is to generalize to novel scenarios. We evaluate the generalizability of our learned policies in two different ways. In the first set of experiments, we collect fresh demonstrations that were not in the training set, and we condition our policies on such trajectories. We find that across the different tasks, even with unseen conditionings, C-BeT retains 67% of the single-task performance, with 16/50 task successes in total. In the second set of experiments, we add environmental distractors in the setup (Figure 4.1, bottom three rows) and run the single- and multi-task conditions on the modified environments. We see once again that

the performance drops to around 67% of original with two distractors on the scene, but if we keep adding (four or more) distractors, the robot is unable to complete any tasks.

4.5.3 Analysis of Failure Modes

We see a few failure modes in our experiments that may provide additional insights into learning from real-world play data. We discuss the most salient ones in this section.

Failure in knob operation in the real world: We see that in all of our real world experiments, the accuracy in operating the knob is consistently lower than all other tasks. This is due to the failure of the learned representations. Upon inspection of the dataset images' nearest neighbors in the representation space, we see that the BYOL-trained representation cannot identify the knob state better than random chance: the returned nearest neighbor differs in knob status often. Since the representation cannot identify the knob status properly, conditioning on it naturally fails.

Importance of a multi-modal policy architecture: One of our motivations behind incorporating the BeT architecture in our work is its ability to learn multi-modal action distributions. In our experiments, we show that for some single-task conditions such as opening the oven door, having no multi-modality is sufficient (Table 4.4), but for more complicated tasks and learning from a more interconnected form of play data, it is always the case that a multi-modal architecture prevents our policies from collapsing to sub-optimal solutions (Table 4.5).

4.6 Related Work

Outcome-conditioned behavior learning: Behavior learning conditioned on particular outcomes, such as reward or goals, is a long studied problem [Kaelbling 1993; Schaul et al. 2015; Veeriah et al. 2018; Zhao et al. 2019]. Compared to standard behavior learning, learning conditioned behavior can generally be more demanding since the same model can be expected to learn

a multitude of behaviors depending on the outcome, which can make learning long-term behavior harder [Levy et al. 2017; Nachum et al. 2018]. As a result, a common line of work in outcome-conditioned learning is to use some form of relabeling of demonstrations or experience buffer as a form of data augmentation [Kaelbling 1993; Andrychowicz et al. 2017; Ghosh et al. 2019; Goyal et al. 2022a] similar to what we do in the paper. As opposed to goal or state conditioned learning, which we focus on in this paper, recently reward conditioned learning using a transformer [Chen et al. 2021] was introduced. However, later work found that it may not work as expected in all environments [Paster et al. 2022; Brandfonbrener et al. 2022] and large transformer models may not be necessary [Emmons et al. 2021] for reward conditioned learning. In this work, we find that using transformers is crucial, particularly when dealing with high dimensional visual observation and multi-modal actions.

Learning from play data: Our work is most closely related to previous works such as Lynch et al. [2020]; Gupta et al. [2019], which also focus on learning from play demonstrations that may not be strictly optimal and uniformly curated for a single task. Learning policies capable of multiple tasks from play data allows knowledge sharing, which is why it may be more efficient compared to learning from demonstrations directly [Zhang et al. 2018b; Rahmatizadeh et al. 2018; Duan et al. 2017; Pari et al. 2021; Young et al. 2021]. [Gupta et al. 2022] attempts reset-free learning with play data, but requires human annotation and instrumentation in the environment for goal labels.

Generative modeling of behavior: Our method of learning a generative model for behavior learning follows a long line of work, including Inverse Reinforcement Learning or IRL [Russell 1998; Ng et al. 2000; Ho and Ermon 2016], where given expert demonstrations, a model tries to construct the reward function, which is then used to generate desirable behavior. Another class of algorithms learn a generative action decoder [Pertsch et al. 2021; Singh et al. 2020] from interaction data to make downstream reinforcement learning faster and easier, nominally making multi-modal action distribution easier. Finally, a class of algorithms, most notably Liu et al. [2020];

Florence et al. [2022]; Kostrikov et al. [2021]; Nachum and Yang [2021] do not directly learn a generative model, but instead learn energy based models that need to be sampled to generate behavior, although they do not primarily focus on goal-conditioning.

Transformers for behavior learning: Our work follows earlier notable works in using transformers to learn a behavior model from an offline dataset, such as [Chen et al. 2021; Janner et al. 2021; Shafiullah et al. 2022]. Our work is most closely related to [Shafiullah et al. 2022] as we build on their transformer architecture, while our unimodal baseline is a variant of [Chen et al. 2021] that learns outcome conditioned instead of reward conditioned policy. Beyond these, [Dasari and Gupta 2020; Mandi et al. 2021] summarizes historical visual context using transformers, and [Clever et al. 2021] relies on the long-term extrapolation abilities of transformers as sequence models. The goal of C-BeT is orthogonal to these use cases, but can be combined with them for future applications.

4.7 Limitations

In this work, we have presented C-BeT, a new approach for conditional behavior generation that can learn from offline play data. Across a variety of benchmarks, both simulated and real, we find that C-BeT significantly improves upon prior state-of-the-art work. However, we have noticed two limitations in C-BeT, particularly for real-robot behavior learning. First, if the features provided to C-BeT do not appropriately capture relevant objects in the scene, the robot execution often fails to interact with that object in its environment. Second, some tasks, like opening the oven door, have simpler underlying data that is not multimodal, which renders only meager gains with C-BeT. A more detailed analysis of these limitations are presented in Section 4.5.3. We believe that future work in visual representation learning can address poor environment features, while the collection of even larger play datasets will provide more realistic offline data for large-scale behavior learning models.

Postscript

C-BeT extends BeT in an intuitive way – by adding a conditioning head into the transformer. It is surprising that this method is still the best way of controlling the behavior of a large behavior model. Recent investigation in such large behavior models are showing that more "advanced" language conditioning methods are acting as a weak signal to the poorly-controllable policy and often like a one-hot or multi-hot encoding conditioning rather than true language understanding and goal directed behavior. One of the potential routes that seem promising in this front is generating "imagined" intermediate states that can then be used as a condition for the policy.

Acknowledgement

This work was led by Jeff Cui, co-authored with Yibin Wang, and advised by Lerrel Pinto. We thank Sridhar Arunachalam, David Brandfonbrener, Irmak Guzey, Yixin Lin, Jyo Pari, Abitha Thankaraj, and Austin Wang for their valuable feedback and discussions. This work was supported by awards from Honda, Meta, Hyundai, Amazon, and ONR award N000142112758.

5 | Behavior Generation with Latent Actions: Vector-Quantized Behavior Transformers

5.1 Introduction

The presently dominant paradigm in modeling human outputs, whether in language [Achiam et al. 2023], image [Podell et al. 2023], audio [Ziv et al. 2024], or video [Bar-Tal et al. 2024], follows a similar recipe: collect a large in-domain dataset, use a large model that fits the dataset, and possibly as a cherry on top, improve the model output using some domain-specific feedback or datasets. However, such a large, successful model for generating human or robot actions in embodied environments has been absent so far, and the issues are apparent. Action sequences are semantically diverse but temporally highly correlated, human behavior distributions are massively multi-modal and noisy, and the hard-and-fast grounding in the laws of physics means that unlike audio, language or video-generation, even the smallest discrepancies may cause a cascade of consequences that lead to catastrophic failures in as few as tens of timesteps [Ross et al. 2011; Rajaraman et al. 2020]. The desiderata for a good model of behaviors and actions thus must contain the following abilities: to model long- and short-term dependencies, to capture and generate from

diverse modes of behavior, and to replicate the learned behaviors precisely [Shafiullah et al. 2022; Chi et al. 2023].

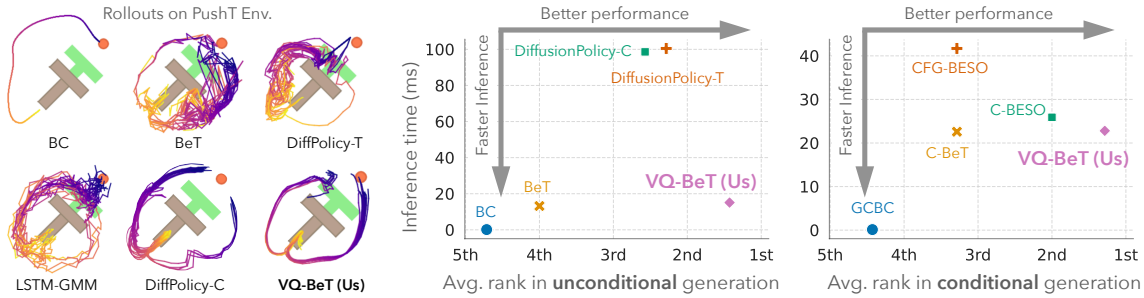


Figure 5.1: Qualitative and quantitative comparison between VQ-BeT and relevant baselines. On the left, we can see trajectories generated by different algorithms while pushing a T-block to target, where VQ-BeT generates smooth trajectories covering both modes. On the right, we show two plots comparing VQ-BeT and relevant baselines on unconditional and goal-conditional behavior generation. The comparison axes are (x-axis) relative success represented by average rank on a suite of seven simulated tasks, and (y-axis) inference time.

Prior work by [Shafiullah et al. 2022] shows how transformers can capture the temporal dependencies well, and to some extent even capture the multi-modality in the data with clever tokenization. However, that tokenziation relies on k-means clustering, a method typically based on an ℓ_2 metric space that unfortunately does not scale to high-dimensional action spaces or temporally extended actions with lots of inter-dependencies. More recent works have also used tools from generative modeling to address the problem of behavior modeling [Pearce et al. 2023; Chi et al. 2023; Zhao et al. 2023b], but issues remain, for example in high computational cost when scaling to long-horizons, or failing to express multi-modality during rollouts.

In this work, we propose Vector Quantized Behavior Transformer (VQ-BeT), which combines the long-horizon modeling capabilities of transformers with the expressiveness of vector-quantization to minimize the compute cost while maintaining high fidelity to the data. We posit that a large part of the difficulty in behavior modeling comes from representing the continuous-valued, multi-modal action vectors. A ready answer is learning discrete representations using vector quantization [Van

Den Oord et al. 2017] used extensively to handle the output spaces in audio [Dhariwal et al. 2020], video [Wu et al. 2021], and image [Rombach et al. 2022]. In particular, the performance of VQ-VAEs for generative tasks has been so strong that a lot of recent models that generate continuous values simply generate a latent vector in the VQ-space first before decoding or upsampling the result [Ziv et al. 2024; Bar-Tal et al. 2024; Podell et al. 2023].

VQ-BeT is designed to be versatile, allowing it to be readily used in both conditional and unconditional generation, while being performative on problems ranging across simulated manipulation, autonomous driving, and real-robotics. Through extensive experiments across eight benchmark environments, we present the following experimental insights:

1. VQ-BeT achieves state-of-the-art (SOTA) performance on unconditional behavior generation outperforming BC, BeT, and diffusion policies in 5/7 environments (Figure 5.1 middle). Quantitative metrics of entropy and qualitative visualizations indicate that this performance gain is due to better capture of multiple modes in behavior data (Figure 5.1 left).

2. On conditional behavior generation, by simply specifying goals as input, VQ-BeT achieves SOTA performance and improves upon GCBC, C-BeT, and BESO in 6/7 environments (Figure 5.1 right).

3. VQ-BeT directly works on autonomous driving benchmarks such as nuScenes [Caesar et al. 2020], matching and being comparable to task-specific SOTA methods.

4. VQ-BeT is a single-pass model, and hence offers a 5× speedup in simulation and 25× on real-world robots over multi-pass models that use diffusion models.

5. VQ-BeT scales to real-world robotic manipulation such as pick-and-placing objects and door closing, improving upon prior work by 73% on long-horizon tasks.

5.2 Background and Preliminaries

5.2.1 Behavior cloning

Given a dataset of continuous-valued action and observation pairs $\mathcal{D} = \{(o_t, a_t)\}_t$, the goal of behavior cloning is to learn a mapping π from observation space $\mathcal{O}$ to the action space $\mathcal{A}$. This map is often learned in a supervised fashion with π as a deep neural network minimizing some loss function $\mathcal{L}(\pi(o), a)$ on the observed behavior data pairs $(o, a) \in \mathcal{D}$. Traditionally, $\mathcal{L}$ was simply taken as the MSE loss, but its inability to admit multiple modes of action for an observation led to different loss formulations [Lynch et al. 2020; Florence et al. 2022; Shafiullah et al. 2022; Chi et al. 2023]. Similarly, understanding that the environment may be partially observable led to modeling the distribution $\mathbb{P}(a_t \mid o_{t-h:t})$ rather than $\mathbb{P}(a_t \mid o_t)$. Finally, understanding that such behavior datasets are often generated with an explicit or implicit goal, many recent approaches condition on an (implicit or explicit) goal variable g and learn a goal-conditioned behavior $\mathbb{P}(a \mid o, g)$. Note that such behavior datasets crucially do not contain any "reward" information, which makes this setup different from reward-conditioned learning as a form of offline RL.

5.2.2 Behavior Transformers

Behavior transformer (BeT) [Shafiullah et al. 2022] and conditional behavior transformer (C-BeT) [Cui et al. 2022] are respectively two unconditional and goal-conditional behavior cloning algorithms built on top of GPT-like transformer architectures. In their respective settings, they have shown the ability to handle temporal correlations in the dataset, as well as the presence of multiple modes in the behavior. While GPT [Brown et al. 2020] itself maps from discrete to discrete domains, BeT can handle multi-modal continuous output space by a clever tokenization trick. Prior to training, BeT learns a k-means based encoder/decoder that can convert continuous actions

into one discrete and one continuous component. Then, by learning a categorical distribution over the discrete component and combining the component mean with a predicted continuous "offset" variable, BeT can functionally learn multiple modes of the data while each mode remains continuous. While the tokenizer allows BeT handle multi-modal actions, the use of k-means means that choosing a good value of k is important for such algorithms. In particular, if k is too small then multiple modes of action gets delegated to the same bin, and if k is too large one mode gets split up into multiple bins, both of which may result in a suboptimal policy. Also, when the action has a large number of (potentially correlated) dimensions, for example when performing action chunking [Zhao et al. 2023b], non-parametric algorithms like k-means may not capture the nuances of the data distribution. Such shortcomings of the tokenizer used in BeT and C-BeT is one of the major inspirations behind our work.

5.2.3 Residual Vector Quantization

In order to tokenize continuous action, we employ Residual Vector Quantization (Residual VQ) [Zeghidour et al. 2021] as a discretization bottleneck. Vector quantization is a quantization technique where continuous values are replaced by a finite number of potentially learned codebook vectors. This process maps the input x to an embedding vector z_q in the codebook $\{e_1, e_2, \cdots e_k\}$ by the nearest neighbor look-up:

$$z_q = e_c, \quad \text{where } c = \text{argmin}_j ||x - e_j||_2. \tag{5.1}$$

Residual VQ is a multi-stage vector quantizer [Vasuki and Vanathi 2006] which replaces each embedding of vanilla VQ-VAE [Van Den Oord et al. 2017] with the sum of vectors from a finite layers of codebooks. This approach cascades N_q layers of vector quantizations residually: the input vector x is passed through the first stage of vector quantization to derive z_q^1. The residual, $x - z_q^1$, is then iteratively quantized by a sequence of $N_q - 1$ quantizing layers, passing the updated

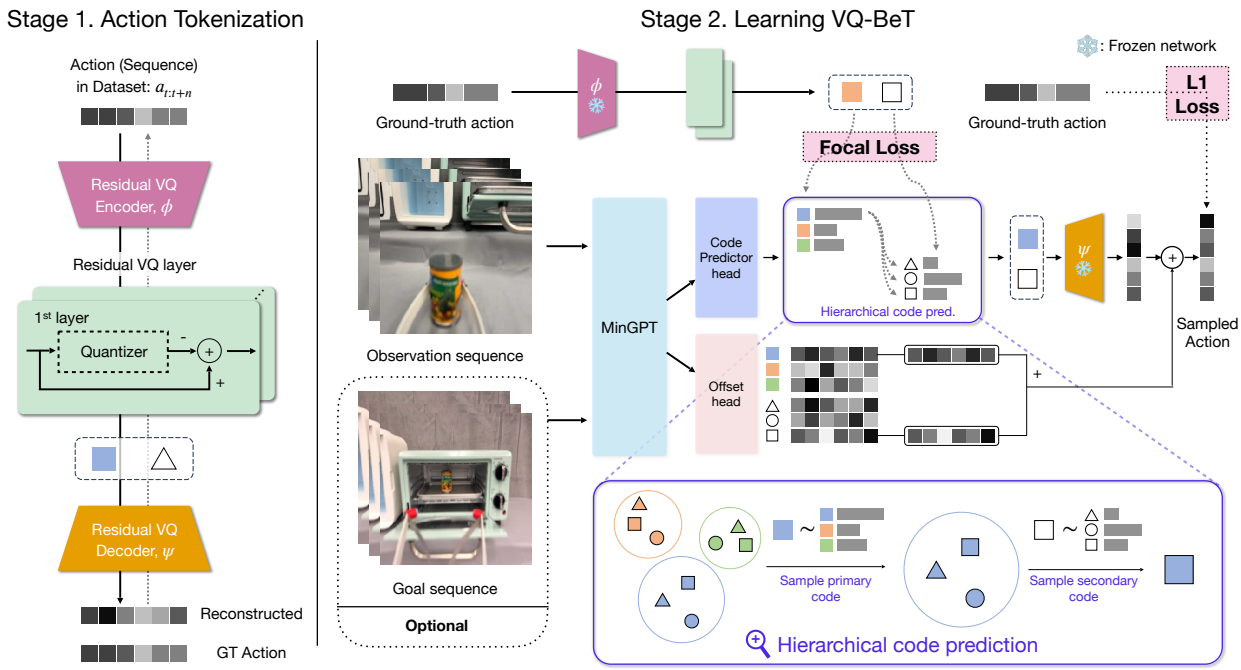


Figure 5.2: Overview of VQ-BeT, broken down into the residual VQ encoder-decoder training phase and the VQ-BeT training phase. The same architecture works for both conditional and unconditional cases with an optional goal input. In the bottom right, we show a detailed view of the hierarchical code prediction method.

residual $x - \sum_{i=1}^{p} z_q^i$ to the next layer. The final quantized input vector is then the sum of vectors from a set of finite codebooks $z_q(x) = \sum_{i=1}^{N_q} z_q^i$.

5.3 Vector-Quantized Behavior Transformers

In this section, we introduce VQ-BeT, which has capability to solve both conditional and non-conditional tasks from uncurated behavior dataset. VQ-BeT is composed of two stages: Action discretization phase (stage 1 in Figure 5.2) and VQ-BeT learning phase (stage 2 in Figure 5.2). Each stage is explained in Section 5.3.2 and 5.3.3, respectively.

5.3.1 Sequential prediction on behavior data

Binning actions to tokenize them and predicting the tokenized class has been successfully applied for learning multi-modal behavior [Shafiullah et al. 2022; Cui et al. 2022]. However, these k-means binning approaches face issues while scaling, as disucssed in Section 5.2.2.

As such, we propose instead to learn a discrete latent embedding space for action or action chunks, and modeling such action latents instead. Note that, such latent models in the form of VQ-VAEs and latent diffusion models are widely used in multiple generative modeling subfields, including image, music, and video [Bar-Tal et al. 2024; Ziv et al. 2024; Podell et al. 2023]. With such discrete tokenziation, our model can directly predict action tokens from observation sequences optionally conditioned on goal vectors.

5.3.2 Action (chunk) discretization via Residual VQ

We employ Residual VQ-VAE [Zeghidour et al. 2021] to learn a scalable action discretizer and address the complexity of action spaces encountered in the real world. The quantization process of an action (or action chunk, where $n > 1$) $a_{t:t+n}$ is learned via learning a pair of encoder and decoder networks; ϕ, ψ. We start with passing $a_{t:t+n}$ through the encoder ϕ. The resulting latent embedding vector $x = \phi(a_{t:t+n})$ is then mapped to an embedding vector in the codebook of the first layer $z_q^1 \in \{e_1^1, \cdots e_k^1\}$ by the nearest neighbor look-up, and the residual is recursively mapped to each codebook of the remaining $N_q - 1$ layers $z_q^i \in \{e_1^i, \cdots e_k^i\}$, where $i = 2, \cdots, N_q$. The latent embedding vector $x = \phi(a_{t:t+n})$ is represented by the sum of vectors from codebooks $z_q(x) = \sum_{i=1}^{N_q} z_q^i$, where each vector $z_q^{i=1:N_q}$ works as the centroid of hierarchical clustering.

Then, the discretized vector $z_q(x) = \sum_{i=1}^{N_q} z_q^i$ is reconstructed as $\psi(z_q(x))$ by passing through the decoder ψ. We train Residual VQ-VAE using a loss function, as shown in Eq 5.3. The first term represents the reconstruction loss, and the second term is the VQ objective that shifts the

embedding vector e towards the encoded action $x = \phi(a_{t:t+n})$. To update the embedding vectors $e_{1:k}^{1:N_q}$, we use moving averages rather than direct gradient updates following [Islam et al. 2022; Mazzaglia et al. 2022]. In all of our experiments, it was sufficient to use $N_q := 2$ VQ-residual layers, and keep the commitment loss $\lambda_{\text{commit}} := 1$ constant.

$$\mathcal{L}_{\text{Recon}} = \left\| a_{t:t+n} - \psi(z_q(\phi(a_{t:t+n}))) \right\|_1 \tag{5.2}$$

$$\mathcal{L}_{\text{RVQ}} = \mathcal{L}_{\text{Recon}} + \|\text{SG}[\phi(a_{t:t+n})] - e\|_2^2 \tag{5.3}$$

$$+\lambda_{\text{commit}} \|\phi(a_{t:t+n}) - \text{SG}[e]\|_2^2, \quad (\text{SG} : \text{stop gradient})$$

We indicate the codes of the first quantizer layer as *primary code*, and the codes of the remaining layers as *secondary codes*. Intuitively, the primary codes in Residual VQ performs coarse clustering over a large range within the dataset, while the secondary codes handle fine-grained actions. (Decoded centroids are visualized in Appendix Figure D.2.)

5.3.3 Weighted update for code prediction

After training Residual VQ, we train GPT-like transformer architecture to model the probability distribution of action or action chunks from the sequence of observations. One of the main differences between BeT and VQ-BeT stems from using a learned latent space. Since our vector quantization codebooks let us freely translate between an action latent $z_q(\phi(a_{t:t+n})) = \sum_{i=1}^{N_q} z_q^i$ and the sequence of chosen codes at each codebook, $\{z_q^i\}_{i=1}^{N_q}$, we use them as a labels in the code prediction $\mathcal{L}_{\text{code}}$ loss to learn the categorical prediction head ζ_{code}^i for given sequence of observations $o_{t-h:t}$. Following [Shafiullah et al. 2022; Cui et al. 2022], we employ Focal loss [Lin et al. 2017] to train the code prediction head by comparing the probabilities of the predicted categorical distribution with the actual labels z_q^i. We adjust the weights between the primary code

and secondary code learning losses, leveraging our priors about the latent space.

$$\mathcal{L}_{\text{code}} = \mathcal{L}_{\text{focal}}(\zeta^{i=1}_{\text{code}}(o_t)) + \beta\mathcal{L}_{\text{focal}}(\zeta^{i>1}_{\text{code}}(o_t)) \tag{5.4}$$

Finally, the quantized behavior is obtained by passing the sum of the predicted residual embeddings through the decoder as follows.

$$\lfloor a_{t:t+n} \rfloor = \psi\Big(\sum_{j,i} e^i_j \cdot \mathbb{I}[\zeta^i_{\text{code}} = j)] \Big) \tag{5.5}$$

We adopt additional offset head ζ_{offset} to maintain full fidelity, adjusting the centers of discretized actions based on observations. The total VQ-BeT loss is shown in Eq. 5.7.

$$\mathcal{L}_{\text{offset}} = \Big| a_{t:t+n} - \big(\lfloor a_{t:t+n} \rfloor + \zeta_{\text{offset}}\,(o_t) \big) \Big|_1 \tag{5.6}$$

$$\mathcal{L}_{\text{ VQ–BeT}} = \mathcal{L}_{\text{code}} + \mathcal{L}_{\text{offset}} \tag{5.7}$$

5.3.4 Conditional and non-conditional task formulation

To provide a general-purpose behavior-learning model that can predict multi-modal continuous actions in both conditional and unconditional tasks, we introduce conditional and non-conditional task formulation of VQ-BeT.

Non-conditional formulation: For a given dataset $\mathcal{D} = \{o_t, a_t\}$, we consider a problem of predicting the distribution of possible action sequences $a_{t:t+n}$ conditioned on a sampled sequence of observations $o_{t-h:t}$. Thus, we formulate the behavior policy as $\pi : \mathcal{O}^h \rightarrow \mathcal{A}^n$, where $\mathcal{O}$ and $\mathcal{A}$ denotes the observation space and action space, respectively.

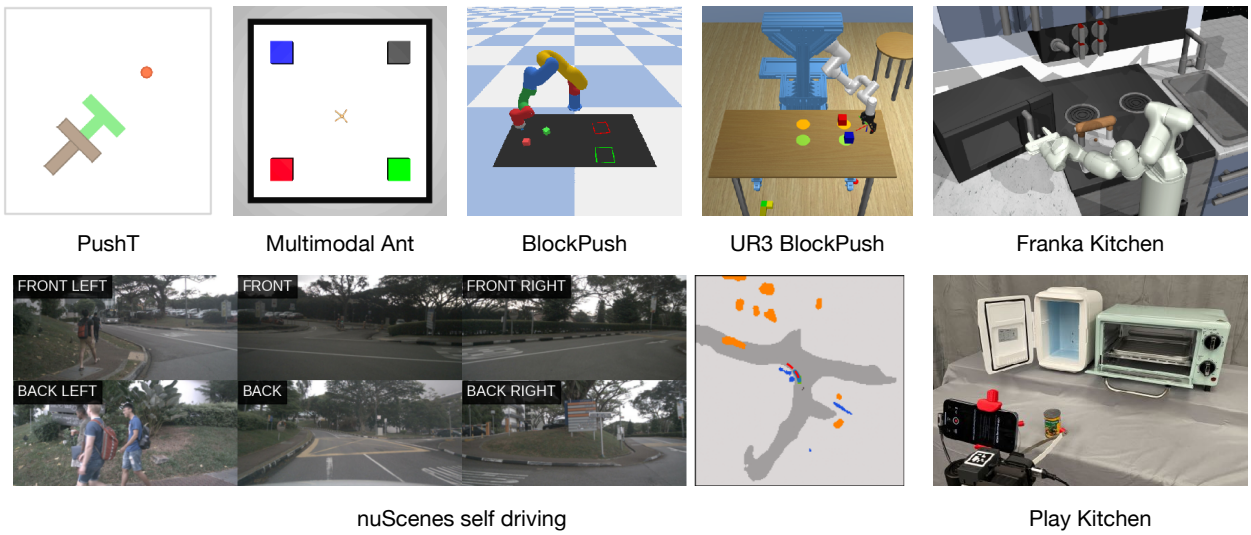


Figure 5.3: Visualization of the environments (simulated and real) where we evaluate VQ-BeT. Top row contains PushT [Chi et al. 2023], Multimodal Ant [Brockman et al. 2016], BlockPush [Florence et al. 2022], UR3 BlockPush [Kim et al. 2022], Franka Kitchen [Gupta et al. 2019], and bottom row contains nuScenes self-driving [Caesar et al. 2020], and our real robot environment.

CONDITIONAL FORMULATION: For goal-conditional tasks, we extend the formulation above to take a goal conditioning vector in the form of one or more observations. Given current observation sequence and future observation sequence, we now consider an extended policy model that predicts the distribution of sequential behavior $\pi : \mathcal{O}^h \times \mathcal{O}^g \rightarrow \mathcal{A}^n$, where $o_{t-h:t} \in \mathcal{O}^h$ and $o_{N-g:N} \in \mathcal{O}^g$ are current and future observation sequences.

5.4 EXPERIMENTS

With both conditional and unconditional VQ-BeT, we run experiments to understand how well they can model behavior on different datasets and environments. We focus on two primary properties of VQ-BeT's generated behaviors: quality, as evaluated by how well the generated behavior achieves some task objective or goal, and the diversity, as evaluated by the entropy of the distribution of accomplished subtasks or goals. Concretely, through our experiments, we try

Environment	Metric	GCBC	C-BeT	C-BESO	CFG-BESO	VQ-BeT
PushT	Final IoU	0.02	0.02	0.30	0.25	**0.39**
Image PushT	(·/1)	0.02	0.01	0.02	0.01	**0.10**
Kitchen	Goals	0.15	3.09	3.75	3.47	**3.78**
Image Kitchen	(·/4)	0.64	2.41	2.00	1.59	**2.60**
Multimodal Ant	Goals	0.00	1.68	1.14	0.92	**1.72**
UR3 BlockPush	(·/2)	0.19	1.67	**1.94**	1.91	**1.94**
BlockPush	Success (·/1)	0.01	0.87	**0.93**	0.88	0.87

Table 5.1: Comparing different algorithms in goal-conditional behavior generation. The seven simulated robotic manipulation and locomotion environments used here are described in Section 5.4.1.

to answer the following questions:

1. How well do VQ-BeT policies perform on the respective environments in both conditional and unconditional behavior generation?
2. How well does VQ-BeT capture the multi-modality present in the dataset?
3. Does VQ-BeT scale beyond simulated tasks?
4. What design choices of VQ-BeT make the most impact in its performance?

5.4.1 Environments, datasets, and baselines

Across our experiments, we use a variety of environments and datasets to evaluate VQ-BeT (Figure 5.3). In simulation, we evaluate the wider applicability of VQ-BeT on eight benchmarks; namely, six manipulation tasks including two image-based tasks: (a) PushT, (b) Image PushT, (c) Kitchen, (d) Image Kitchen, (e) UR3 BlockPush, (f) BlockPush; a locomotion task, (g) Multimodal Ant; and a self-driving benchmark, (h) NuScenes. The environments are visualized in Figure 5.3, and a detailed descriptions of each task is provided in Appendix D.1.1. We also evaluate on a real-world environment with twelve tasks (five single-phase, three multi-phase tasks and four long-horizon tasks) described in Section 5.4.7.

Environment	Diffusion Policy	VQ-BeT
PushT	0.73	**0.78**
Image PushT	0.66	**0.68**
Kitchen	3.44	**3.66**
Image Kitchen	**3.11**	2.98
Multimodal Ant	3.12	**3.22**
UR3 BlockPush	1.83	**1.84**
BlockPush	**1.93**	1.79
Real kitchen (1 task)	0.9	**0.94**
Real kitchen (2 tasks)	0.37	**0.63**

Table 5.2: Performance of different algorithms in unconditional behavior generation tasks. We evaluate over seven simulated robotic manipulation and locomotion tasks as described in Section 5.4.1.

Baselines: We compare VQ-BeT against the SOTA methods in behavior modeling in both conditional and unconditional categories. In both of these categories, we compare against transformer- and diffusion-based baselines.

For unconditional behavior generation, we compare against MLP-based behavior cloning, the original Behavior Transformers (BeT) [Shafiullah et al. 2022] and Diffusion Policy [Chi et al. 2023]. The BeT architecture uses a k-means tokenization as explained in Section 5.2.2. Diffusion policy [Chi et al. 2023], on the other hand, uses a denoising diffusion head [Ho et al. 2020] to model multi-modality in the behaviors. We use both the convolutional and transformer variant of the diffusion policy as baselines for our work since they excel in different cases.

For goal-conditional behaviors, we compare against simple goal conditioned BC, Conditional Behavior Transformers (C-BeT) [Cui et al. 2022] and BESO [Reuss et al. 2023]. C-BeT uses k-means tokenization but otherwise has a similar architecture to ours. BESO uses denoising diffusion, and has a conditioned variant (C-BESO) and a classifier-free guided variant (CFG-BESO) that we compare against.

5.4.2 Performance of behavior generated by VQ-BeT

We evaluate VQ-BeT in a set of goal-conditional tasks in Table 5.1 and a set of unconditional tasks in Table 5.2. On the PushT environments, we look at final and max coverage, where the coverage value is the IoU between the T block and the target T position. For the unconditional Kitchen, BlockPush, and Ant tasks, we look at the total number of tasks completed in expectation, where the maximum possible number of tasks is 4, 2, and 4 respectively. For the conditional environments, we report the expected number of successes given a commanded goal sequence, where the numbers of commanded goals are 4 in Kitchen, 2 in Ant, and 2 in BlockPush. Across all of these metrics, a higher number designates a better performance.

From Tables 5.1 and 5.2, we see that in both conditional and unconditional tasks, VQ-BeT largely outperforms or matches the baselines. First, on the conditional tasks, we find that VQ-BeT outperforms all baselines in all tasks except for BlockPush. In BlockPush, VQ-BeT performs on par with BeT, while C-BESO and CFG-BESO performs slighly better. Note that BlockPush has one of the simplest action spaces (2-D $\Delta x, \Delta y$) in the dataset while also having the largest demonstration dataset, and thus the added advantage of having vector quantized actions may not have such a strong edge. Next, in unconditional tasks, we find that VQ-BeT outperforms all baselines in Franka Kitchen (state), Ant Multimodal, UR3 Multimodal, and both PushT (state and image) environments. In BlockPush environment, VQ-BeT is outperformed by DiffusionPolicy-T, while in Image Kitchen it is outperformed by DiffusionPolicy-C. However, VQ-BeT empirically shows stable performances on all tasks, while DiffusionPolicy-T struggles in Image PushT environments, and DiffusionPolicy-C underperforms in Kitchen and BlockPush environments.

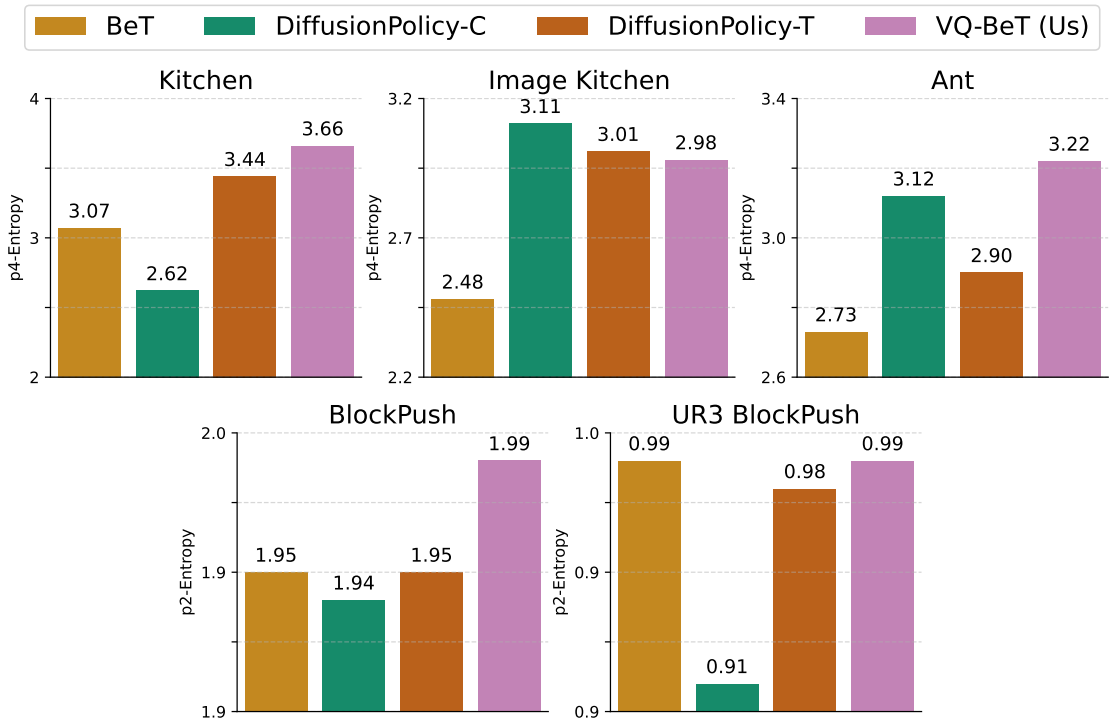


Figure 5.4: A comparison between the behavior entropy of the algorithms, calculated based on their task completion order, on five of our simulated environments.

5.4.3 How well does VQ-BeT capture multimodality?

One of the primary promises of behavior generation models is to capture the diversity present in the data, rather than simply copying a single mode of the existing data very well. Thus, for a quantitative measure we examine the behavior entropy of the models in the unconditional behavior generation task. Behavior entropy here tries to captures the diversity of a model's generated long horizon behaviors. We compare the final-subtask entropy as a balanced metric between performance and diversity. We see that VQ-BeT outperforms all baselines in all tasks except for Image Kitchen, where it's outperformed by DiffusionPolicy-T. However, behavior diversity is hard to capture properly in a single number, which is why we also present the diversity of generated behavior on the PushT task in Figure 5.1 (left). There, we can see how VQ-BeT captures both

modes of the dataset in rollouts, while also generating overall smooth trajectories.

5.4.4 Inference-time efficiency of VQ-BeT

Unconditional	C-BeT	C-BESO	CFG-BESO	VQ-BeT
Single step	22.6ms	25.9ms	41.7ms	22.8ms
Multi step	✗	✗	✗	23.3ms
Conditional	**BeT**	**DiffusionPolicy-C**	**DiffusionPolicy-T**	**VQ-BeT**
Single step	13.2ms	100.5ms	98.6ms	15.1ms
Multi step	✗	100.7ms	98.6ms	15.2ms

Table 5.3: Inference times for VQ-BeT and baselines in kitchen environment. For DiffusionPolicy we rolled-out with 10-iteration diffusion, following their real-world settings. The methods that only support single-step action prediction are marked with ✗.

Denoising diffusion based models such as DiffusionPolicy and BESO require multiple forward passes from the network to generate a single action or action chunk. In contrast, VQ-BeT can generate action or action chunks in a single forward pass. As a result, VQ-BeT enjoys much faster inference times, as shown in Table 5.3. Receding horizon control using action chunking can speed up some of our baselines, but VQ-BeT can take advantage of the same, speeding up the method proportionally. Moreover, receding horizon control is not a silver bullet; it can be problematic in affordable, inaccurate hardware, as we show in Section 5.4.7 in our real world experiments.

5.4.5 Adapting VQ-BeT for autonomous driving

While our previous experiments showed robotic manipulation or locomotion results, learning from multi-modal behavior datasets has wider applications. We evaluate VQ-BeT in one such case, in a self-driving trajectory planning task using the nuScenes [Caesar et al. 2020] dataset. In this task, given a few frames of observations, the model must predict the next six frames of an car's location. While nuScenes usually require the trajectory be predicted from the raw images, we

Method	Access to information	Avg. L_2 (m) (↓)	Avg. collision (%) (↓)
FF [Hu et al. 2021]	Full	1.43	0.43
EO [Khurana et al. 2022]		1.6	0.33
UniAD [Hu et al. 2023]		1.03	0.31
Agent-Driver [Mao et al. 2023b]		<u>0.74</u>	<u>0.21</u>
GPT-Driver [Mao et al. 2023a]	Partial	0.84	0.44
Diffusion-based traj. model		0.96	0.49
VQ-BeT		**0.73**	**0.29**

Table 5.4: *(Lower is better)* Trajectory planning performance on the nuScenes environment. We **bold** the best partial-information model and <u>underline</u> the best full-information model. Even with partial information about the environment, VQ-BeT can match or beat the SOTA models on the L_2 error metric.

adapted the GPT-Driver [Mao et al. 2023a] framework which uses pretrained models to extract vehicle and obstacle locations and velocities. However, this processing also discards road lane and shoulder informations, which makes collision avoidance hard.

In Table 5.4, we show the performance of VQ-BeT in this task, measured by how closely it followed the ground truth trajectory in test scenes, as well as how likely the generated trajectory was to collide with the environment. Note that collision avoidance is especially difficult for agents with partial information since they do not have any lane information. We find that VQ-BeT outperforms all other methods in trajectory following, achieving the lowest average L_2 distance between the ground truth trajectories and generated trajectories. Moreover, VQ-BeT achieves a collision probability that is better or on-par with older self-driving methods, while not being designed for self-driving in particular.

5.4.6 Design decisions that matter for VQ-BeT

In this section, we examine how changes in each module of VQ-BeT affect its performance. We ablate the following components: using residual vs. vanilla VQ, using an offset head, using action chunking, predicting the VQ-codes autoregressively, and weighing primary and secondary codes equally by setting $\beta = 1$ in Eq. 5.4. We perform these ablation experiments in the condi-

tional Kitchen, unconditional Ant, and the nuScenes self-driving task, and the result summary is presented in Figure 5.5.

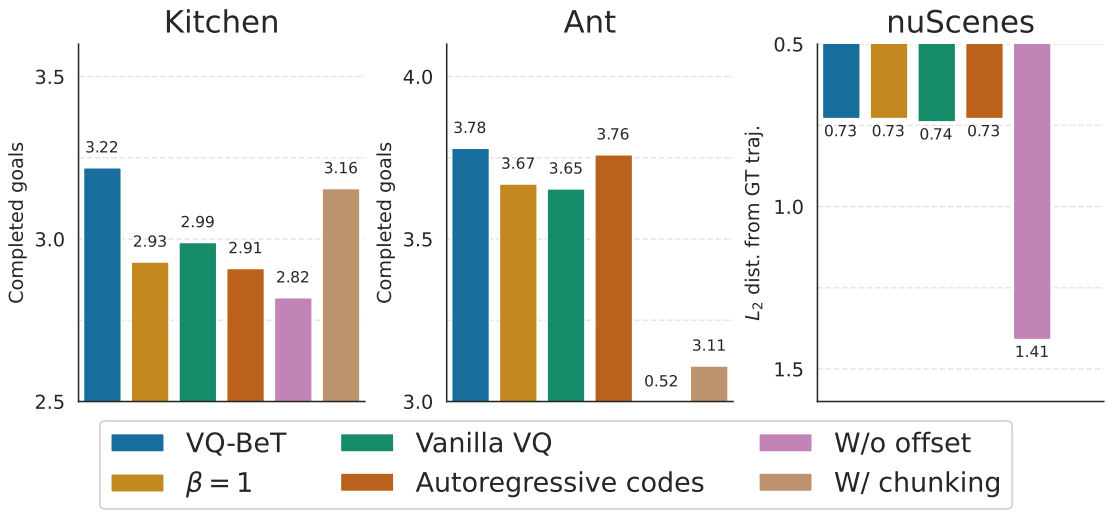


Figure 5.5: Summary of our ablation experiments. The five different axes of ablation is described in Section 5.4.6.

We note that performance-wise, not using a residual VQ layer has a significant negative impact, which we believe is because of the lack of expressivity from a single VQ-layer. A similar drop in performance shows up when we weigh the two VQ layers equally by setting $\beta = 1$, in Eq. 5.4. Both experiments seems to provide evidence that important expressivity is conferred on VQ-BeT using residual VQs. Next, we note that predicting the VQ-codes autoregressively has a negative impact on the kitchen environment. This performance drop is anomalous, since in the real world, we found that the autoregressive (and thus causal) prediction of primary and secondary codes is important for good performance. In the environments where it is possible, we also tried action chunking [Zhao et al. 2023b]; however the performance for such models were lacking. Since VQ-BeT models are small and fast, action chunking isn't necessary even when running it on a real robot in real time. Finally, we found that the offset prediction is quite important for VQ-BeT, which points to how important full action fidelity is for sequential decision making tasks that we

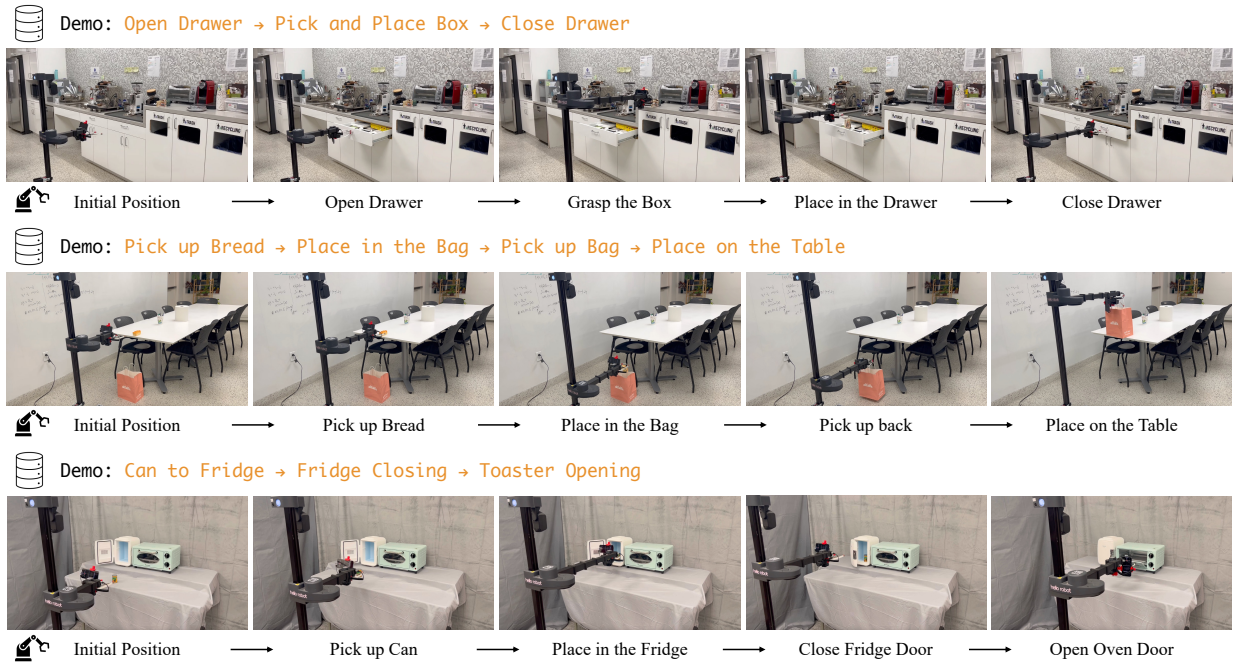


Figure 5.6: Visualization of the trajectory VQ-BET generated in a long-horizon real world environment. Each demo consists of three to four consecutive tasks. Please refer to Table 5.6 for the success rates for each task.

evaluate on.

5.4.7 Adapting VQ-BeT to real-world robots

While our previous experiments evaluated VQ-BeT in simulated environments, one of the primary potential applications of it is in learning robot policies from human demonstrations. In this section, we set up a real robot environment, collect some data, and evaluate policies learned using VQ-BeT.

Environment and dataset: For single-phase and two-phase tasks, we run our experiments in a kitchen-like environment with a toaster oven, a mini-fridge, and a small can in front of the robot as shown in Figure 5.3. For long-horizon scenarios consisting of more than three tasks, we also test on a real kitchen environment as shown in Figure 5.6. We use a similar robot and data collection

setup as Dobb·E [Shafiullah et al. 2023b], and use the Hello Robot: Stretch [Kemp et al. 2022] for policy rollouts. We create a set of single-phase and multi-phase tasks on this environment (See Table 5.5, or Appendix D.1.2 for details). While the single-phase tasks can only be completed in one way, some multi-phase tasks have multi-modal solutions in the benchmark and the datasets.

Baselines: In this environment, we use MLP-BC and BC with Depth as our simple baselines, and DiffusionPolicy-T as our multi-modal baseline. To handle visual inputs, all models are prepended with the HPR encoder from Shafiullah et al. [2023b] which is then fine-tuned during training.

Method	Open Toaster	Close Toaster	Close Fridge	Can to Toaster	Can to Fridge	Total
VQ-BeT	**8/10**	**10/10**	**10/10**	**10/10**	9/10	**47/50**
DiffPol-T†	**8/10**	9/10	8/10	**10/10**	**10/10**	45/50
BC w/ Depth	0/10	7/10	**10/10**	8/10	2/10	27/50
BC	0/10	8/10	7/10	9/10	5/10	29/50

Method	Can to Fridge → Close Fridge	Can to Toaster → Close Toaster	Close Fridge and Toaster	Total
VQ-BeT	**6/10**	**8/10**	5/10	**19/30**
DiffPol-T†	4/10	1/10	**6/10**	11/30
BC w/ Depth	2/10	0/10	2/10	4/30
BC	2/10	1/10	4/10	7/30

Table 5.5: Real world robot experiments solving a number of standalone tasks (top) and two-task sequences (bottom). Here, † denotes that we modified DiffusionPolicy-T to improve its performance; see Section 5.4.7 paragraph "Practical concerns".

Results: We present the experiment results from the real world environment in Table 5.5 and Table 5.6. Table 5.5 is split in two halves for single-phase and two-phase tasks. On the single-phase tasks, we see that, simple MLP-BC models are able to perform almost all tasks with some success, which shows that the subtasks are achievable, and the baselines are implemented well. On these single-phase tasks, VQ-BeT marginally outperforms DiffusionPolicy-T, while both algorithms achieve a $\geq$ 90% success rate. However, the more interesting comparison is in the two-phase,

longer horizon tasks. Here, VQ-BeT outperforms all baselines, including DiffusionPolicy, by a relative margin of 73%.

Besides comparisons with baselines, we also notice multimodality in the behavior of VQ-BeT. Especially in the task "Close Fridge and Toaster", we note that our model closes the doors in both possible orders during rollouts rather than collapsing to a single mode of behavior.

Task 1	Approach Handle	Grasp Handle	Open Drawer	Let Handle Go	Approach the Box	Grasp the Box	Move to Drawer	Place Box inside	Go in front of Drawer	Close Drawer
VQ-BeT	8/10	7/10	7/10	7/10	7/10	7/10	7/10	6/10	6/10	6/10
DiffPol-T†	10/10	9/10	9/10	9/10	8/10	3/10	3/10	3/10	3/10	2/10
Task 2	Approach Bread	Grasp the Bread	Move to the Bag	Place Bread inside	Approach the Handle	Grasp the Handle	Lift Bag up	Place on the table	Let Handle go	
VQ-BeT	10/10	10/10	10/10	4/10	3/10	3/10	3/10	3/10	3/10	
DiffPol-T†	9/10	9/10	9/10	9/10	2/10	2/10	2/10	1/10	1/10	
Task 3	Grasp Can	Pick up Can	Can into Fridge	Let Go of Can	Move Left of Fridge Door	Close Fridge Door	Go in Front of Toaster	Grasp Toaster Handle	Open Toaster	Return to Home Pos.
VQ-BeT	10/10	10/10	10/10	8/10	8/10	8/10	8/10	7/10	7/10	7/10
DiffPol-T†	5/10	5/10	5/10	4/10	2/10	2/10	2/10	2/10	2/10	2/10
Task 4	Grasp Can	Pick up Can	Can into Toaster	Drops Can on Tray	Goes Below Toaster Door	Close Toaster Door	Backs up	Move Left of Fridge Door	Close Fridge	Return to Home Pos.
VQ-BeT	10/10	10/10	8/10	8/10	8/10	6/10	6/10	6/10	6/10	6/10
DiffPol-T†	9/10	9/10	8/10	8/10	8/10	1/10	2/10	2/10	2/10	1/10

Table 5.6: Long-horizon real world robot experiments (Figure 5.6). Each task consists of three to four sequences; Task 1 (Open Drawer → Pick and Place Box → Close Drawer), Task 2 (Pick up Bread → Place in the Bag→ Pick up Bag → Place on the Table), Task 3 (Can to Fridge → Fridge Closing → Toaster Opening), and Task 4 (Can to Toaster → Toaster Closing → Fridge Closing). Here, † denotes that we modified DiffusionPolicy-T to improve its performance as explained in Section 5.4.7 paragraph "Practical concerns".

Additionally, we present results from long-horizon real world experiments consisting of a sequence of three or more subtasks in Figure 5.6 and Table 5.6. We consider interactions with a wider variety of environments (communal kitchen and conference room) and objects (bread, box, bag, and drawer) compared to the single- or two-phase tasks in order to evaluate VQ-BeT in more general scenes. Overall, we see that VQ-BeT has at least thrice the success rate of DiffusionPolicy at the end of all four tasks. For Task 1 and 2, we observe that VQ-BeT gains a performance advantage toward the end of the episode, although VQ-BeT and DiffusionPolicy perform similarly at the beginning of the episodes. Also note that Task 2 is difficult in our ego-only camera setup, since the bag is out of the view while grabbing the bread. For Tasks 3 and 4, we observe that VQ-BeT outperforms DiffusionPolicy in all subtasks and notably, the performance difference is even more pronounced toward the end of the episode. These long-horizon task results continue to suggest that VQ-BeT may overfit less and learn more robust behavior policies in longer horizons tasks.

	RTX A4000	Intel i3 CPU
VQ-BeT	**18.06**	**207.25**
Diffusion Policy	573.49	5243.82

Table 5.7: Average inference time for real robot (in milliseconds). The GPU column is calculated on our workstation while the CPU column is calculated on the Hello Robot's onboard computer.

Practical concerns: In practice, we noticed that receding-horizon control as used by Chi et al. [2023] fails completely in our environment (See Appendix Table D.4 for comparison to closed loop control). Our low-cost mobile manipulator robot lacks precise motion control unlike more expensive robot arms like Franka Panda. This controller noise causes models to go out of distribution during even a short period (three timesteps) of open-loop rollout. To resolve this, we rolled out every policy fully closed-loop, which resulted in a much larger inference time gap (25×) between VQ-BeT and Diffusion Policy as presented in Table 5.7.

5.5 Related Works

Deep generative models for modeling behavior: VQ-BeT builds on a long line of works that leveraged tools from generative modeling to learn diverse behaviors. The earliest examples are in inverse RL literature [Kalakrishnan et al. 2013; Wulfmeier et al. 2015; Finn et al. 2016; Ho and Ermon 2016], where such tools were used to learn a reward function given example behavior. Using generative priors for action generationi, such as GMM by Lynch et al. [2020] or EBMs by Florence et al. [2022], or simply fitting multi-modal action distributions [Singh et al. 2020; Pertsch et al. 2021] became more common with large, human collected behavior datasets [Mandlekar et al. 2018; Gupta et al. 2019]. Subsequently, a large body of work [Shafiullah et al. 2022; Cui et al. 2022; Pearce et al. 2023; Chi et al. 2023; Reuss et al. 2023; Chen et al. 2023] used generative modeling tools for generalized behavior learning from multi-modal datasets.

Action reparametrization: While Shafiullah et al. [2022] is the closest analogue to VQ-BeT, the practice of reparametrizing actions for easier or better control goes back to "bang-bang" controllers [Bushaw 1952; Bellman et al. 1956] replacing continuous actions with extreme discrete values. Discretizing each action dimension separately, however, may exponentially explode the action space, which is generally addressed by assuming each action dimension as independent [Tavakoli et al. 2018] or causally dependent [Metz et al. 2017]. Without priors on the action space, each of these assumptions may be limiting, which is why later work opted to learn the reparametrization [Singh et al. 2020; Dadashi et al. 2021; Luo et al. 2023] similar to VQ-BeT. On another hand, options [Sutton et al. 1999; Stolle and Precup 2002] abstract actions temporally but can be challenging to learn from data. Many applications instead hand-craft primitives as a parametrized action space [Hausknecht and Stone 2015] which may not scale well for different tasks.

5.6 Limitations

In this work, we introduce VQ-BeT, a model for learning behavior from open-ended, multi-modal data by tokenizing the action space using a residual VQ-VAE, and then using a transformer model to predict the action tokens. While we show that VQ-BeT performs well on a plethora of manipulation, locomotion, and self-driving tasks, an exciting application of such models would be in scaling them up to large behavior datasets containing orders of magnitude more data, environments, and behavior modes. Finding a shared latent space of actions between different embodiments may let us "translate" policies between different robots or even from human to robots. Finally, a learned, discrete action space may also make real-world RL application faster, which we would like to explore in the future.

Postscripts

VQ-BeT completes the arc of BeT by creating a significantly more mature action representation. Using a learned action representation was something BeT did, but the learning algorithm matters a lot as shown in this work. There are natural extensions possible, such as action chunking and Fourier-space representations, and could easily be integrated to this line of work. However, we show that finding compact, discrete representation of the actions is critical for learning small and fast policies.

One of the ways in which the BeT line of work is still underappreciated is their speed and the size of the models. Even with many experiments, we have not been able to produce similarly small and fast models using other algorithms, which will be very relevant for future applications in robots deployed in the fields.

Acknowledgements

This work was led by Seungjae (Jay) Lee, co-authored with Yibin Wang, Haritheja Etukuru, H. Jin Kim, and co-advised with Lerrel Pinto. NYU authors are supported by grants from Amazon, Honda, and ONR award numbers N00014-21-1-2404 and N00014-21-1-2758. This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [NO.2021-0-01343, Artificial Intelligence Graduate School Program (Seoul National University)]. NMS is supported by the Apple Scholar in AI/ML Fellowship. LP is supported by the Packard Fellowship. We thank Jonghae Park for his help in obtaining the UR3 Multimodal dataset.

Part II

Mechanisms for Generalizable Scaling

6 | On Bringing Robots Home with Hardware and Efficient Algorithms

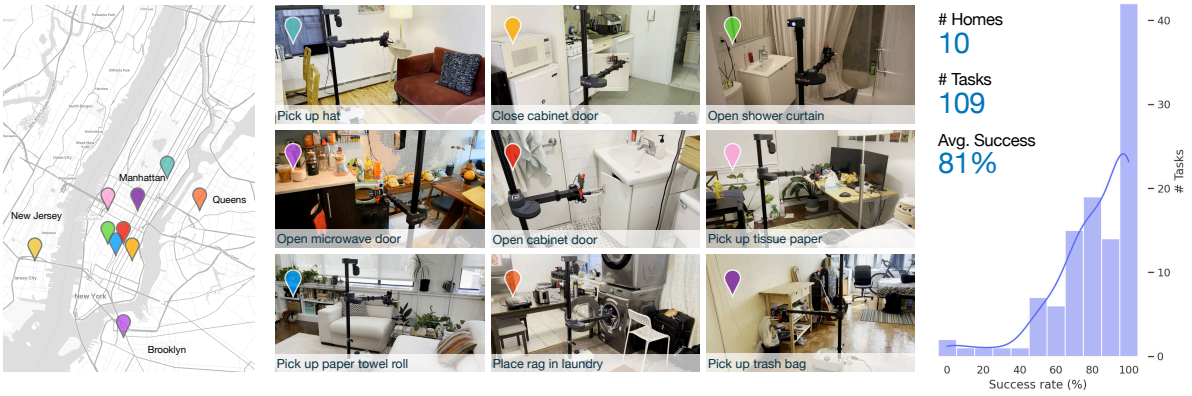


Figure 6.1: We present Dobb·E, a simple framework to train robots, which is then field tested in homes across New York City. In under 30 mins of training per task, Dobb·E achieves 81% success rates on simple household tasks.

6.1 Introduction

Since our transition away from a nomadic lifestyle, homes have been a cornerstone of human existence. Technological advancements have made domestic life more comfortable, through innovations ranging from simple utilities like water heaters to advanced smart-home systems. However, a holistic, automated home assistant remains elusive, even with significant representations in popular culture [Carper 2019].

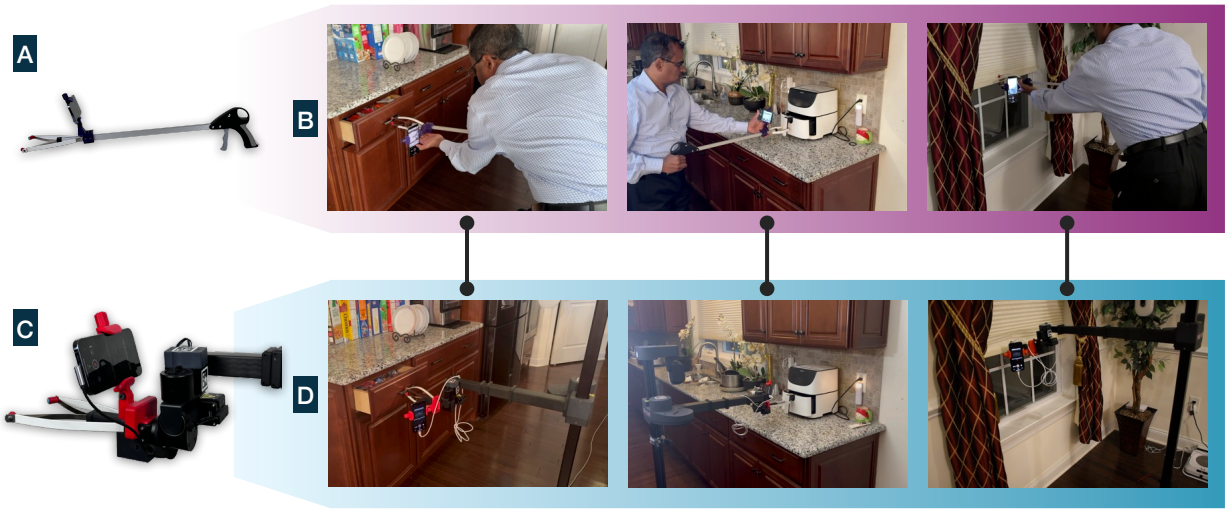


Figure 6.2: (A) We design a new imitation learning framework, starting with a data collection tool. (B) Using this data collection tool, users can easily collect demonstrations for household tasks. (C) Using a similar setup on a robot, (D) we can transfer those demos using behavior cloning techniques to real homes.

Our goal is to build robots that perform a wide-range of simple domestic tasks across diverse real-world households. Such an effort requires a shift from the prevailing paradigm – current research in robotics is predominantly either conducted in industrial environments or in academic labs, both containing curated objects, scenes, and even lighting conditions. In fact, even for the simple task of object picking [Gupta et al. 2018] or point navigation [Gervet et al. 2023a] performance of robotic algorithms in homes is far below the performance of their lab counterparts. If we seek to build robotic systems that can solve harder, general-purpose tasks, we will need to reevaluate many of the foundational assumptions in lab robotics.

In this work we present Dobb·E, a framework for teaching robots in homes by embodying three core principles: efficiency, safety, and user comfort. For efficiency, we embrace large-scale data coupled with modern machine learning tools. For safety, when presented with a new task, instead of trial-and-error learning, our robot learns from a handful of human demonstrations. For user comfort, we have developed an ergonomic demonstration collection tool, enabling us to gather task-specific demonstrations in unfamiliar homes without direct robot operation.

Figure 6.3: We ran experiments in a total of 10 homes near the New York City area, and successfully completed 102 out of 109 tasks that we tried. The figure shows a subset of 60 tasks, 6 tasks from 10 homes each, from our home robot experiments using Dobb·E.

Concretely, the key components of Dobb·E include:

- **Hardware**: The primary interface is our demonstration collection tool, termed the "Stick." It combines an affordable reacher-grabber with 3D printed components and an iPhone. Additionally, an iPhone mount on the robot facilitates direct data transfer from the Stick without needing domain adaptation.
- **Pretraining Dataset**: Leveraging the Stick, we amass a 13 hour dataset called Homes of New York (HoNY), comprising 5620 demonstrations from 216 environments in 22 New York homes, bolstering our system's adaptability. This dataset serves to pretrain representation models for Dobb·E.
- **Models and algorithms**: Given the pretraining dataset we train a streamlined vision model, called Home Pretrained Representations (HPR), employing cutting-edge self-supervised learning (SSL) techniques. For novel tasks, a mere 24 demonstrations sufficed to finetune this vision model, incorporating both visual and depth information to account for 3D reasoning.
- **Integration**: Our holistic system, encapsulating hardware, models, and algorithms, is centered around a commercially available mobile robot: Hello Robot Stretch [Kemp et al. 2022].

We run Dobb·E across 10 homes spanning 30 days of experimentation, over which it tried 109 tasks and successfully learned 102 tasks with performance $\geq 50\%$ and an overall success rate of 81%. Concurrently, extensive experiments run in our lab reveals the importance of many key design decisions. Our key experimental findings are:

- **Surprising effectiveness of simple methods:** Dobb·E follows a simple behavior cloning recipe for visual imitation learning using a ResNet model [He et al. 2016] for visual representation extraction and a two-layer neural network [Rosenblatt 1958] for action prediction (see Section 6.2). On average, only using 91 seconds of data on each task collected over five minutes, Dobb·E can achieve a 81% success rate in homes (see Section 6.3).

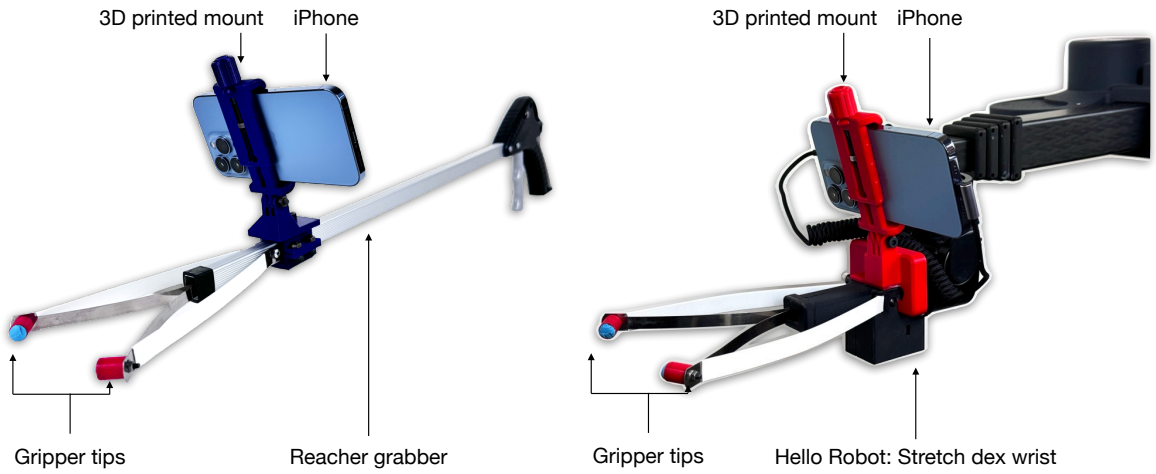
- **Impact of effective SSL pretraining:** Our foundational vision model, HPR trained on home data improves tasks success rate by at least 23% compared to other foundational vision models [Xiao et al. 2022; Nair et al. 2022b; Majumdar et al. 2023], which were trained on much larger internet datasets (see Section 6.3.4.1).
- **Odometry, depth, and expertise:** The success of Dobb·E is heavily reliant on the Stick providing highly accurate odometry and actions from the iPhones' pose and position sensing, and depth information from the iPhone's Lidar. Ease of collecting demonsrations also makes iterating on research problems with the Stick much faster and easier (see Section 6.3.4).
- **Remaining challenges:** Hardware constraints such as the robot's force, reach, and battery life, limit tasks our robot can physically solve (see Section 6.3.3.3), while our policy framework suffers with ambiguous sensing and more complex, temporally-extended tasks (see Sections 6.3.3.4, 6.4.1).

To encourage and support future work in home robotics, we have open-sourced our code, data, models, hardware designs, and are committed to supporting reproduction of our results. More information along with robot videos are available on our project website: `https://dobb-e.com`.

6.2 Technical Components and Method

To create Dobb·E we partly build new robotic systems from first principles and partly integrate state-of-the-art techniques. In this section we will describe the key technical components in Dobb·E. To aid in reproduction of Dobb·E, we have open sourced all of the necessary ingredients in our work; please see Section 6.5 for more detail.

At a high level, Dobb·E is an behavior cloning framework [Atkeson and Schaal 1997]. Behavior cloning is a subclass of imitation learning, which is a machine learning approach where a model



(A) The Stick (B) Hello Robot: Stretch setup

Figure 6.4: Photographs of our designed hardware, including the (A) Stick and the (B) identical iPhone mount for Hello Robot: Stretch wrist. From the iPhone's point of view, the grippers look identical between the two setups.

learns to perform a task by observing and imitating the actions and behaviors of humans or other expert agents. Behavior cloning involves training a model to mimic a demonstrated behavior or action, often through the use of labeled training data mapping observations to desired actions. In our approach, we pretrain a lightweight foundational vision model on a dataset of household demonstrations, and then in a new home, given a new task, we collect a handful of demonstrations and fine-tune our model to solve that task. However, there are many aspects of behavior cloning that we created from scratch or re-engineered from existing solutions to conform to our requirements of efficiency, safety, and user comfort.

Our method can be divided into four broad stages: (a) designing a hardware setup that helps us in the collection of demonstrations and their seamless transfer to the robot embodiment, (b) collecting data using our hardware setup in diverse households, (c) pretraining foundational models on this data, and (d) deploying our trained models into homes.

6.2.1 Hardware Design

The first step in scaling robotic imitation to arbitrary households requires us to take a closer look at the standard imitation learning process and its inefficiencies. Two of the primary inefficiencies in current real-world imitation learning lay in the process of collecting the robotic demonstrations and transferring them across environments.

6.2.1.1 Collecting robot demonstrations

The standard approach to collect robot demonstrations in a robotic setup is to instrument the robot to pair it with some sort of remote controller device [Mandlekar et al. 2018; Arunachalam et al. 2023a], a full robotic exoskeleton [Fang et al. 2023a; Falck et al. 2019; Zhao et al. 2023a; Ishiguro et al. 2020], or simpler data collection tools [Song et al. 2020; Young et al. 2020; Pari et al. 2021]. Many recent works have used a video game controller or a phone [Mandlekar et al. 2018], RGB-D cameras [Arunachalam et al. 2023b], or virtual reality device [Arunachalam et al. 2023a; Guzey et al. 2023b,a] to control the robot. Other works [Zhao et al. 2023b] have used two paired robots in a scene where one of the robots is physically moved by the demonstrator while the other robot is recorded by the cameras. However, such approaches are hard to scale up to households efficiently. Physically moving a robot is generally unwieldy, and for a home robotic task would require having multiple robots present at the site. Similarly, full exoskeleton based setups as shown in [Fang et al. 2023a; Zhao et al. 2023a; Ishiguro et al. 2020] are also unwieldy in a household setting. Generally, the hardware controller approach suffers from inefficiency because the human demonstrators have to map the controller input to the robot motion. Using phones or virtual reality devices are more efficient, since they can map the demonstrators' movements directly to the robot. However, augmenting these controllers with force feedback is nearly impossible, often leading users to inadvertently apply extra force or torque on the robot. Such demonstrations frequently end up being unsafe, and the generally accepted solution to this problem is to limit the force and torque

users can apply; however, this often causes the robot to diverge from the human behavior.

In this project, we take a different approach by trying to combine the versatility of mobile controllers with the intuitiveness of physically moving the robot. Instead of having the users move the entire robot, we created a facsimile of the Hello Robot Stretch end-effector using a cheap $25 reacher-grabber stick that can be readily bought online, and augmented it ourselves with a 3D printed iPhone mount. We call this tool the "Stick," which is a natural evolution of tools used in prior work [Young et al. 2021; Pari et al. 2021] (see Figure 6.4).

The Stick helps the user intuitively adapt to the limitations of the robot, for example by making it difficult to apply large amounts of force. Moreover, the iPhone Pro (version 12 or newer), with its camera setup and internal gyroscope, allows the Stick to collect RGB image and depth data at 30 frames per second, and its 6D position (translation and rotation). In the rest of the paper, for brevity, we will refer to the iPhone Pro (12 or later) simply as iPhone.

6.2.1.2 Captured Data Modalities

Our Stick collects the demonstration data via the mounted iPhone using an off-the-shelf app called Record3D. The Record3D app is able to save the RGB data at 1280×720 pixels recorded from the camera, the depth data at 256×192 pixels from the lidar sensor, and the 6D relative translation and rotation data from the iPhone's internal odometry and gyroscope. We record this data at 30 FPS onto the phone and later export and process it.

6.2.1.3 Robot Platform

All of our systems are deployed on the Hello Robot Stretch, which is a single-arm mobile manipulator robot already available for purchase on the open market. We use the Stretch RE1 version in all of our experiments, with the dexterous wrist attachment that confers 6D movement abilities on the robot. We chose this robot because it is cheap, lightweight–weighing just 51 pounds (23

kilograms)–and can run on a battery for up to two hours. Additionally, Stretch RE1 has an Intel NUC computer on-board which can run a learned policy at 30 Hz.

6.2.1.4 Camera Mounts

We create and use matching mounts on the Stick and the Hello Robot arm to mount our iPhone, which serves as the camera and the sensor in both cases. One of the main advantages of collecting our data using this setup is that, from the camera's point of view, the Stick gripper and the robot gripper looks identical, and thus the collected data and any trained representations and policies on such data can be directly transferred from the Stick to the robot. Moreover, since our setup operates with only one robot mounted camera, we don't have to worry about having and calibrating a third-person, environment mounted camera, which makes our setup robust to general camera calibration issues and mounting-related environmental changes.

6.2.1.5 Gripper Tips

As a minor modification to the standard reacher-grabber as well as the Hello Robot Stretch end-effector, we replace the padded, suction-cup style tips of the grippers with small, cylindrical tips. This replacement helps our system manipulate finer objects, such as door and drawer handles, without getting stuck or blocked. In some preliminary experiments, we find that our cylindrical tips are better at such manipulations, albeit making pick-and-place like tasks slightly harder.

6.2.2 Pretraining Dataset – Homes of New York

With our hardware setup, collecting demonstrations for various household tasks becomes as simple as bringing the Stick home, attaching an iPhone to it, and doing whatever the demonstrator wants to do while recording with the Record3D app. To understand the effectiveness of the Stick

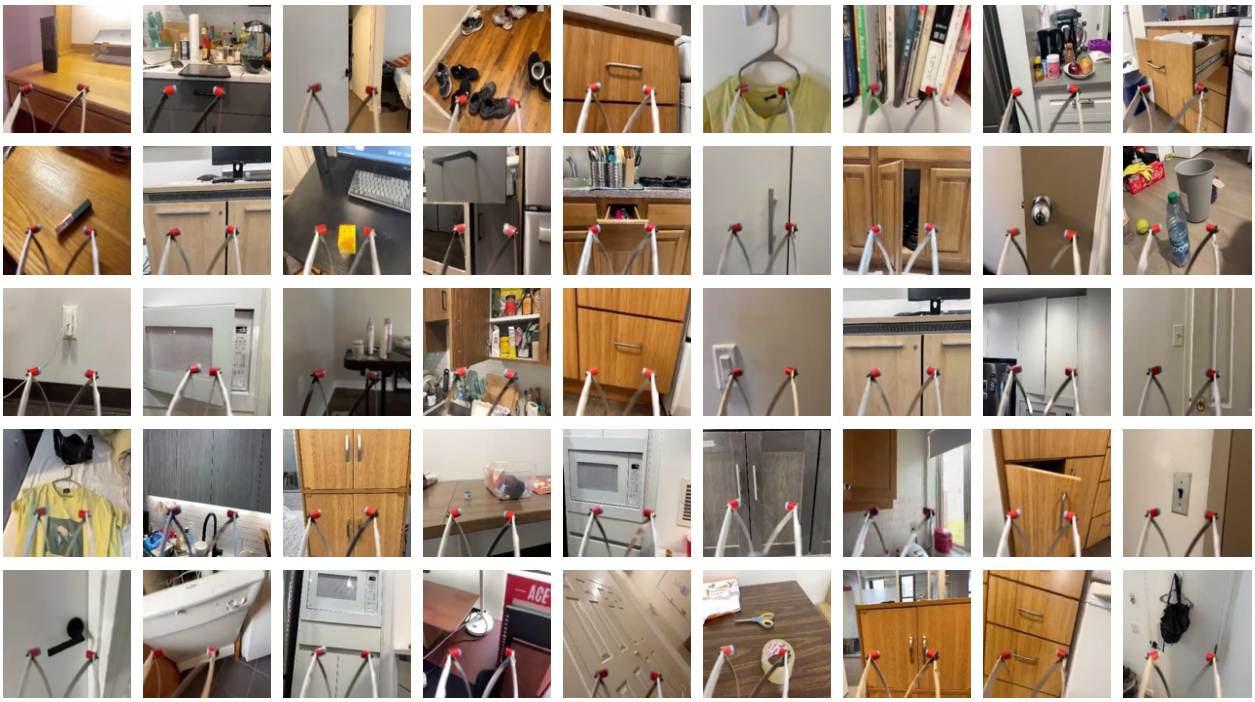


Figure 6.5: Subsample of 45 frames from Homes of New York dataset, collected using our Stick in 22 homes.

as a data collection tool and give us a launching pad for our large-scale learning approach, we, with the help of some volunteers, collected a household tasks dataset that we call Homes of New York (HoNY).

The HoNY dataset is collected with the help of volunteers across 22 different homes, and it contains 5620 demonstrations in 13 hours of total recording time and totalling almost 1.5 million frames. We asked the volunteers to focus on eight total defined broad classes of tasks: switching button, door opening, door closing, drawer opening, drawer closing, pick and place, handle grasping, and play data. For the play data, we asked the volunteers to collect data from doing anything arbitrary around their household that they would like to do using the stick. Such playful behavior has in the past proven promising for representation learning purposes [Young et al. 2021; Guzey et al. 2023b].

We instructed our volunteers to spend roughly 10 minutes to collect demonstrations in each

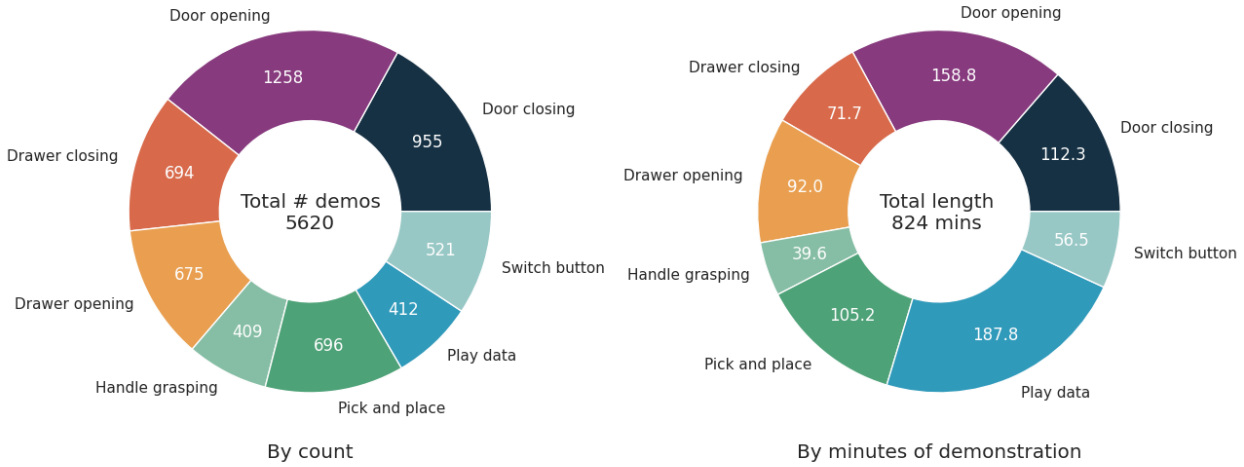


Figure 6.6: Breakdown of Homes of New York dataset by task: on the left, the statistics is shown by number of demonstrations, and on the right, the breakdown is shown by minutes of demonstration data collected.

"environment" or scene in their household. However, we did not impose any limits on how many different tasks they can collect in each home, nor how different each "environment" needs to be across tasks. Our initial demonstration tasks were chosen to be diverse and moderately challenging while still being possible for the robot.

In Figure 6.6, we can see a breakdown of the dataset by the number of frames belonging to each broad class of tasks. As we can see, while there is some imbalance between the number of frames in each task, they are approximately balanced.

Moreover, our dataset contains a mixture of a diverse number of homes, as shown in Figure 6.7, with each home containing 67K frames and 255 trajectories on average.

6.2.2.1 Gripper Data

While the iPhone can give us the pose of the end-effector, there is no way to trivially get the open or closed status of the gripper itself. To address this, we trained a model to track the gripper

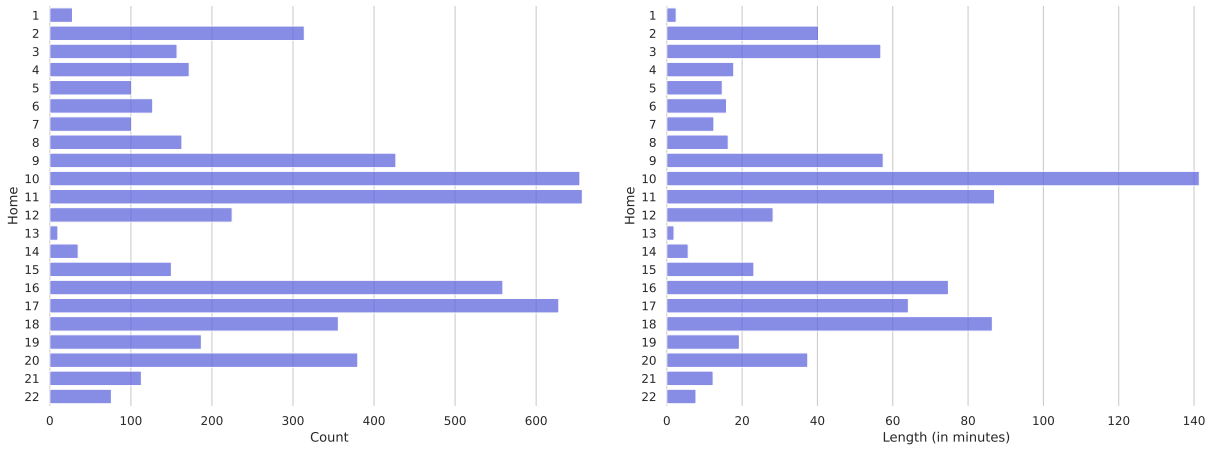


Figure 6.7: Breakdown of our collected dataset by homes. On the left, the statistics are shown by number of demonstrations, and on the right, the breakdown is shown by minutes of demonstration data collected. The Y-axis is marked with the home ID.

tips. We extracted 500 random frames from the dataset and marked the two gripper tip positions in pixel coordinates on those frames. We trained a gripper model on that dataset, which is a 3-layer ConvNet that tries to predict the distance between the gripper tips as a normalized number between 0 and 1. This model, which gets a 0.035 MSE validation error (on a scale from 0-1) on a heldout evaluation set, is then used to label the rest of the frames in the dataset with a gripper value between 0 and 1.

6.2.2.2 Dataset Format

As mentioned in the previous section, we collect the RGB and depth data from the demonstration, as well as the 6D motion of the stick, at 30 Hz. For use in our models, we scale and reshape our images and depths into 256×256 pixels. For the actions, we store the absolute 6D poses of the iPhone at 30 Hz. During model training or fine-tuning, we calculate the relative pose change as the action at the desired frequency during runtime.

6.2.2.3 Dataset Quality Control

We manually reviewed the videos in the dataset to validate them and filter them for any bad demonstrations, noisy actions, and any identifying or personal information. We filtered out any videos that were recorded in the wrong orientation, as well as any videos that had anyone's face or fingers appearing in them.

6.2.2.4 Related Work

Collecting large robotic manipulation datasets is new. Especially in recent years, there have been a few significant advances in collecting large datasets for robotics [Brohan et al. 2023a; Jang et al. 2021; Fang et al. 2023b; Mandlekar et al. 2019; Ebert et al. 2022; Walke et al. 2023; Gupta et al. 2018; Jiang et al. 2011; Pinto and Gupta 2016; Kappler et al. 2015; Mahler et al. 2017a; Depierre et al. 2018; Levine et al. 2018; Kalashnikov et al. 2018; Brahmbhatt et al. 2019; Fang et al. 2020; Eppner et al. 2021; Bousmalis et al. 2018; Zhu et al. 2023; Yu et al. 2016; Finn and Levine 2017; Ebert et al. 2018; Dasari et al. 2019; Kalashnikov et al. 2021; Ebert et al. 2022; Mandlekar et al. 2021; Zitkovich et al. 2023; Lynch et al. 2023; Bharadhwaj et al. 2023; Heo et al. 2023]. While our dataset is not as large as the largest of them, it is is unique in a few different ways. Primarily, our dataset is focused on household interactions, containing 22 households, while most datasets previously were collected in laboratory settings. Secondly, we collect first-person robotic interactions, and are thus inherently more robust to camera calibration issues which affect previous datasets [Sharma et al. 2018; Mandlekar et al. 2018; Cabi et al. 2019; Kalashnikov et al. 2021; Jang et al. 2021; Bharadhwaj et al. 2023]. Thirdly, using an iPhone gives us an advantage over previous work that used cheap handheld tools to collect data [Song et al. 2020; Young et al. 2020; Pari et al. 2021] since we can extract high quality action information quite effortlessly using the onboard gyroscope. Moreover, we collect and release high quality depth information from our iPhone, which is generally rare for standard robotic datasets. The primary reason behind collecting our own dataset instead of

Table 6.1: While previous datasets focused on the number of manipulation trajectories, we instead focus on diverse scenes and environments. As a result, we end up with a dataset that is much richer in interaction diversity.

Dataset	# Traj.	# Env.	# Homes	Public Data	Public Robot	Collection
MIME [Sharma et al. 2018]	8.30k	1	-	✓	✓	human
RoboTurk [Mandlekar et al. 2018]	2.10k	1	-	✓	✓	human
Learning in Homes [Gupta et al. 2018]	28k	9	9	✓	✓	scripted
MT-Opt [Kalashnikov et al. 2021]	800k	1	-	✗	✓	scripted & learned
BC-Z [Jang et al. 2021]	26.0k	1	-	✓	✗	human
RT-1 [Brohan et al. 2023a]	130k	3	-	✓	✗	human
RH20T [Fang et al. 2023b]	110k	50	10	✓	✓	human
RoboSet [Bharadhwaj et al. 2023]	98.5k	11	-	✓	✓	scripted & human
BridgeData v2 [Walke et al. 2023]	60.1k	24	-	✓	✓	human & scripted
HoNY (Us)	5.6k	**216**	**22**	✓	✓	human

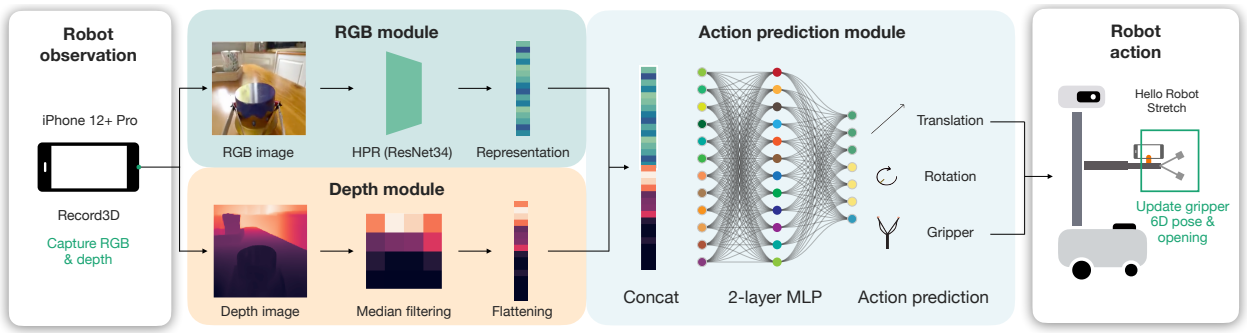


Figure 6.8: Fine-tuning the pretrained HPR model to learn a model that maps from the robot's RGB and depth observations into robot actions: 6D relative pose and the gripper opening.

using any previous dataset is because we believe in-domain pretraining to be a key ingredient for generalizable representations, which we empirically verify in section 6.3.4.1 by comparing with previously released general-purpose robotic manipulation focused representation models. A line of work that may aid in future versions of this work are collections of first-person non-robot household videos, such as [Damen et al. 2018; Grauman et al. 2022; Somasundaram et al. 2023], where they can complement our dataset by augmenting it with off-domain information.

6.2.3 Policy Learning with Home Pretrained Representations

With the diverse home dataset, our next step in the process is to train a foundational visual imitation model that we can easily modify and deploy in homes. To keep our search space small, in this work we only consider simple visual imitation learning algorithms that only consider a single step at a time. While this inevitably limits the capabilities of our system, we leave temporally extended policies as a future direction we want to explore on home robots. Our policy is built of two simple components: a visual encoder and a policy head.

6.2.3.1 Visual Encoder Learning

We use a ResNet34 architecture as a base for our primary visual encoder. While there are other novel architectures that were developed since ResNet34, it satisfies our need for being performant while also being small enough to run on the robot's onboard computer. We pretrain our visual encoder on our collected dataset with the MoCo-v3 self-supervised learning algorithm for 60 epochs. We call this model the Home Pretrained Representation (HPR) model, based on which all of our deployed policies are trained. We compare the effects of using our own visual encoder vs. a pretrained visual encoder trained on different datasets and algorithms, such as R3M [Nair et al. 2022b], VC1 [Majumdar et al. 2023], and MVP [Xiao et al. 2022], or even only pretraining on ImageNet-1K [Deng et al. 2009], in Section 6.3.4.1.

6.2.3.2 Downstream Policy Learning

On every new task, we learn a simple manipulation policy based on our visual encoder and the captured depth values. For the policy, the input space is an RGB-D image (4 channels) with shape 256×256 pixels, and the output space is a 7-dimensional vector, where the first 3 dimensions are relative translations, next 3 dimensions are relative rotations (in axis angle representation), and

the final dimension is a gripper value between 0 and 1. Our policy is learned to predict an action at 3.75 Hz, since that is the frequency with which we subsample our trajectories.

The policy architecture simply consists of our visual representation model applied to the RGB channels in parallel to a median-pooling applied on the depth channel, followed by two fully connected layers that project the 512 dimensional image representation and 512 dimensional depth values down to 7 dimensional actions. During this supervised training period where the network learns to map from observation to actions, we do not freeze any of the parameters, and train them for 50 epochs with a learning rate of 3×10^{-5}. We train our network with a mean-squared error (MSE) loss, and normalize the actions per axis to have zero mean and unit standard deviation before calculating the loss.

Our pretrained visual encoders and code for training a new policy on your own data is available open-source with a permissive license. Please see Section 6.5 for more details.

6.2.3.3 Related Work

While the pretraining-finetuning framework has been quite familiar in other areas of Machine Learning such as Natural Language [Devlin et al. 2018; Brown et al. 2020] and Computer Vision [He et al. 2016; Oquab et al. 2023], it has not caught on in robot learning as strongly. Generally, pretraining has taken the form of either learning a visual representation [Brandfonbrener et al. 2023; Nair et al. 2022b; Majumdar et al. 2023; Xiao et al. 2022; Pari et al. 2021; Young et al. 2021; Radosavovic et al. 2022; Ma et al. 2022a; Karamcheti et al. 2023; Mu et al. 2023; Bahl et al. 2023] or learning a Q-function [Kumar et al. 2022; Herzog et al. 2023] which is then used to figure out the best behavior policy. In this work, we follow the first approach, and pretrain a visual representation that we fine-tune during deployment. While there are recent large-scale robotic policy learning approaches [Brohan et al. 2023a; Zitkovich et al. 2023; Padalkar et al. 2023], the evaluation setup for such policies generally have some overlap with the (pre-)training data. This

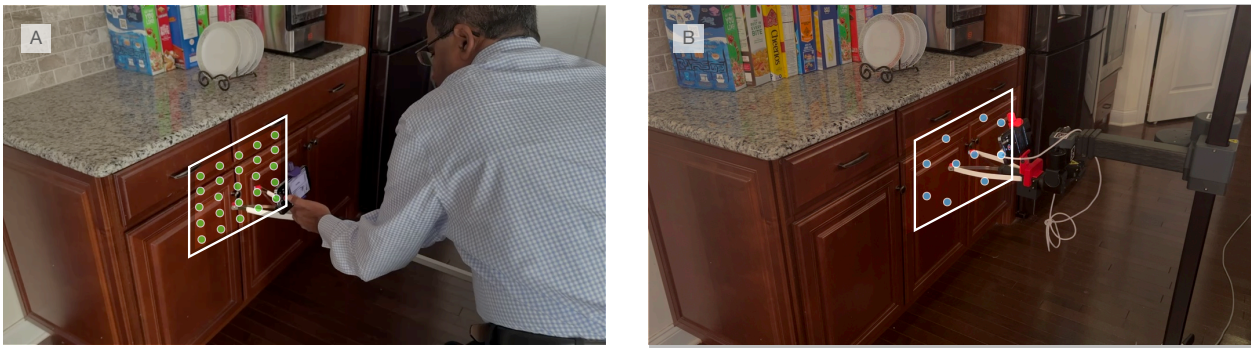


Figure 6.9: (a) The data collection grid: the demonstrator generally started data collection from a 5×5 or 4×6 grid of starting positions to ensure diversity of the collected demos. (b) To ensure our policies generalize to different starting positions, we start the robot policy roll-outs from 10 pre-scheduled starting positions.

work, in contrast, focuses on entirely new households which were never seen during pretraining.

6.2.4 Deployment in Homes

Once we have our Stick to collect data, the dataset preparation script, and the algorithm to fine-tune our pretrained model, the final step is to combine them and deploy them on a real robot in a home environment. In this work, we focus on solving tasks that mostly involve manipulating the environment, and thus we assume that the robot has already navigated to the task space and is starting while facing the task target (which for example could be an appliance to open or an object to manipulate).

6.2.4.1 Protocol for Solving Home Tasks

In a novel home, to solve a novel task, we start by simply collecting a handful of demonstrations on the task. We generally collect 24 new demonstrations as a rule of thumb, which our experiments show is sufficient for simple, five second tasks. In practice, collecting these demos takes us about five minutes. However, some environments take longer to reset, in which case collecting demonstrations may also take longer. To confer some spatial generalization abilities to our robot

policy, we generally collect the data starting from a variety of positions in front of the task setup, generally in a small 4×6 or 5×5 grid (Figure 6.9).

6.2.4.2 Policy Training Details

Once the data is collected, it takes about 5 minutes to process the data from R3D files into our dataset format. From there, for 50 epochs of training it takes about 20 minutes on average on a modern GPU (RTX A4000). As a result, on average, within 30 minutes from the start of the data collection, we end up with a policy that we can deploy on the robot.

6.2.4.3 Robot Execution Details

We deploy the policy on the robot by running it on the robot's onboard Intel NUC computer. We use the iPhone mounted on the arm and the Record3D app to stream RGB-D images via USB to the robot computer. We run our policy on the input images and depth to get the predicted action. We use a PyKDL based inverse kinematics solver to execute the predicted relative action on the robot end-effector. Since the model predicts the motion in the camera frame, we added a joint in the robot's URDF for the attached camera, and so we can directly execute the predicted action without exactly calculating the transform from the camera frame to the robot end-effector frame. For the gripper closing, we binarize the predicted gripper value by applying a threshold that can vary between tasks. We run the policy synchronously on the robot by taking in an observation, commanding the robot to execute the policy-predicted action, and waiting until robot completes the action to take in the next observation. For our evaluation experiments we generally use 10 initial starting positions for each robot task (Figure 6.9 (b)). These starting positions vary our robot gripper's starting position in the vertical and horizontal directions. Between each of these 10 trials, we manually reset the robot and the environment.

6.2.4.4 Related Work

While the primary focus of our work is deploying robots in homes, we are not the first one to do so. The most popular case would be commercial robots such as Roomba [Jones 2006] from iRobot or Astro [Dempsey 2023] from Amazon. While impressive as a commercial product, such closed-source robots are not conducive to scientific inquiry and are difficult to build upon as a community. Some application of robots in home includes early works such as [Nguyen and Kemp 2014] exploring applications of predefined behaviors in homes, [Bhattacharjee et al. 2016, 2018] exploring tactile perception in homes, or [Gupta et al. 2018] exploring the divergence between home and lab data. More recently, ObjectNav, i.e. navigating to objects in the real world [Gervet et al. 2023a] has been studied by taking robots to six different houses. While [Gervet et al. 2023a] mostly experimented on short-term rental apartments and houses, we focused on homes that are currently lived in where cluttered scenes are much more common. There have been other works such as [Bahl et al. 2022; Shah and Levine 2022] which focus on "in the wild" evaluation. However, evaluation-wise, such works have been limited to labs and educational institutions [Bahl et al. 2022], or have focused on literal "in the wild" setups such as cross-country navigation [Shah and Levine 2022].

6.3 Experiments

We experimentally validated our setup by evaluating it across 10 households in the New York and New Jersey area on a total of 109 tasks. On these 109 tasks, the robot gets an 81% success rate, and can complete 102 tasks with at least even odds. Alongside these household experiments, we also set up a "home" area in our lab, with a benchmark suite with 10 tasks that we use to run our baselines and ablations. Note that none of our experiments overlapped with the environments on which our HoNY dataset was collected to ensure that the experimental environments are novel.

6.3.1 List of Tasks in Homes

In Table 6.2 we provide an overview of the 109 tasks that we attempted in the 10 homes, as well as the associated success rate on those tasks. Video of all 109 tasks can also be found on our website: https://dobb-e.com/#videos.

Table 6.2: A list of all tasks in the home enviroments, along with their categories and success rates out of 10 trials.

ID	Home	Task Description	Success ·/10	Task Category
1	1	Door closing: Brown Cabinet	10	Door closing
2	1	Drawer closing: Brown Drawer	10	Drawer closing
3	1	Drawer Opening: Brown Drawer	10	Drawer opening
4	1	Pick up: Plastic Plate	9	Misc object pickup
5	1	Pick up: Flowers	3	Misc object pickup
6	1	Pick and Place: Spices	6	6D pick & place
7	1	Pouring: translucent cup + marshmallows	10	Pouring
8	1	Air Fryer Opening	10	Air-fryer opening
9	1	Air Fryer Closing	10	Air-fryer closing
10	1	Knob Turning	8	Knob turning
11	1	Vertical Blinds Opening	2	Random
12	1	Horizontal Blinds Opening	10	Random
13	2	Sideways washing machine door	8	Door opening
14	2	Dresser drawer	8	Drawer opening
15	2	Placing a rag in laundry	7	6D pick & place
16	2	Picking and placing a keyring	9	6D pick & place
17	2	Pouring: transparent cup	5	Pouring

Continued on the next page

ID	Home	Task Description	Success ·/10	Task Category
18	2	Trash pickup	9	Bag pickup
19	2	Toilet paper unloading	8	Random
20	2	Toaster button pressing	1	Random
21	3	Dishwasher drawer opening	8	Drawer opening
22	3	Cat massager pick and place (onto book)	7	6D pick & place
23	3	Rattatoullie pick and place	5	6D pick & place
24	3	Air fryer opening	0	Air-fryer opening
25	3	Air fryer closing	10	Air-fryer closing
26	3	Chair pulling	10	Chair pulling
27	3	Light switch new demos	8	Light switch
28	3	Unplugging	10	Unplugging
29	3	Towel pickup	7	Towel pickup
30	3	Kettle switch	0	Random
31	3	Shower curtains	6	Random
32	4	Cabinet door closing	10	Door closing
33	4	Closet door opening	7	Door opening
34	4	Freezer door opening	9	Door opening
35	4	Dishwasher door opening	7	Door opening
36	4	Drawer closing	10	Drawer closing
37	4	Hammerhead shark pick and place	4	6D pick & place
38	4	Oil pouring	5	Pouring
39	4	Almonds pouring	6	Pouring
40	4	Chair pulling	8	Chair pulling
41	4	Book pulling	10	Pulling from shelf

Continued on the next page

ID	Home	Task Description	Success ·/10	Task Category
42	4	Tissue pulling	5	Tissue pickup
43	4	Paper bag pickup	8	Bag pickup
44	5	Microwave Door Opening	7	Door opening
45	5	Drawer closing	10	Drawer closing
46	5	Drawer opening	10	Drawer opening
47	5	Chair pulling	10	Chair pulling
48	5	Towel pulling from the fridge	7	Towel pickup
49	5	DVD pulling	10	Pulling from shelf
50	5	Knob turning	5	Knob turning
51	5	Paper towel tube	5	Paper towel replacing
52	6	Door opening kitchen	10	Door opening
53	6	Door opening bathroom	7	Door opening
54	6	Drawer closing	10	Drawer closing
55	6	Mini drawer closing	10	Drawer closing
56	6	Dishwasher drawer opening	8	Drawer opening
57	6	Lantern pick and place	9	6D pick & place
58	6	Chair pulling	10	Chair pulling
59	6	Table pulling	10	Chair pulling
60	6	Rag pull	9	Towel pickup
61	6	Book pulling	8	Pulling from shelf
62	6	Tissue pick up	10	Tissue pickup
63	6	Bag pick up	8	Bag pickup
64	6	Cushion lifting	10	Cushion flipping
65	7	Kitchen door closing	10	Door closing

Continued on the next page

ID	Home	Task Description	Success ·/10	Task Category
66	7	Bathroom closet door opening	9	Door opening
67	7	Drawer closing black wardrode	7	Drawer closing
68	7	Drawer closing white wardrode	10	Drawer closing
69	7	Drawer closing desk	8	Drawer closing
70	7	Drawer closing table	8	Drawer closing
71	7	Chair pulling	9	Chair pulling
72	7	Dining table chair pulling	5	Chair pulling
73	7	Rag pulling	8	Towel pickup
74	7	Tissue paper pick up	10	Tissue pickup
75	7	Paper Towel pick up	10	Paper towel replacing
76	7	Trash pickup	8	Bag pickup
77	8	Door opening	8	Door opening
78	8	Air fryer open	9	Air-fryer opening
79	8	Air fryer close	10	Air-fryer closing
80	8	Chair pulling	10	Chair pulling
81	8	Unplugging	6	Unplugging
82	8	Toilet rag pulling	9	Towel pickup
83	8	Book pulling	8	Pulling from shelf
84	8	Codenames pulling	7	Pulling from shelf
85	8	Tissue pick up	7	Tissue pickup
86	8	Paper towel roll pickup	7	Paper towel replacing
87	8	Food bag pick up	8	Bag pickup
88	8	Cushion flip	10	Cushion flipping
89	8	Toilet flushing	9	Random

Continued on the next page

ID	Home	Task Description	Success ·/10	Task Category
90	9	Door closing	10	Door closing
91	9	Door opening	7	Door opening
92	9	Bathroom drawer closing	10	Drawer closing
93	9	Kitchen drawer closing	10	Drawer closing
94	9	Kitchen drawer opening	6	Drawer opening
95	9	Hat pickup	9	Misc object pickup
96	9	Chair pulling	9	Chair pulling
97	9	Light switch	6	Light switch
98	9	Rag pulling	10	Towel pickup
99	9	Book pulling	7	Pulling from shelf
100	9	Paper bag pick up	10	Bag pickup
101	10	Door Closing	10	Door closing
102	10	Drawer Closing	10	Drawer closing
103	10	Air fryer opening	10	Air-fryer opening
104	10	Air fryer closing	10	Air-fryer closing
105	10	Light switch	8	Light switch
106	10	Hand towel (rag) pulling	7	Towel pickup
107	10	Book pulling	10	Pulling from shelf
108	10	Paper towel	9	Paper towel replacing
109	10	Cushion straightening	10	Cushion flipping

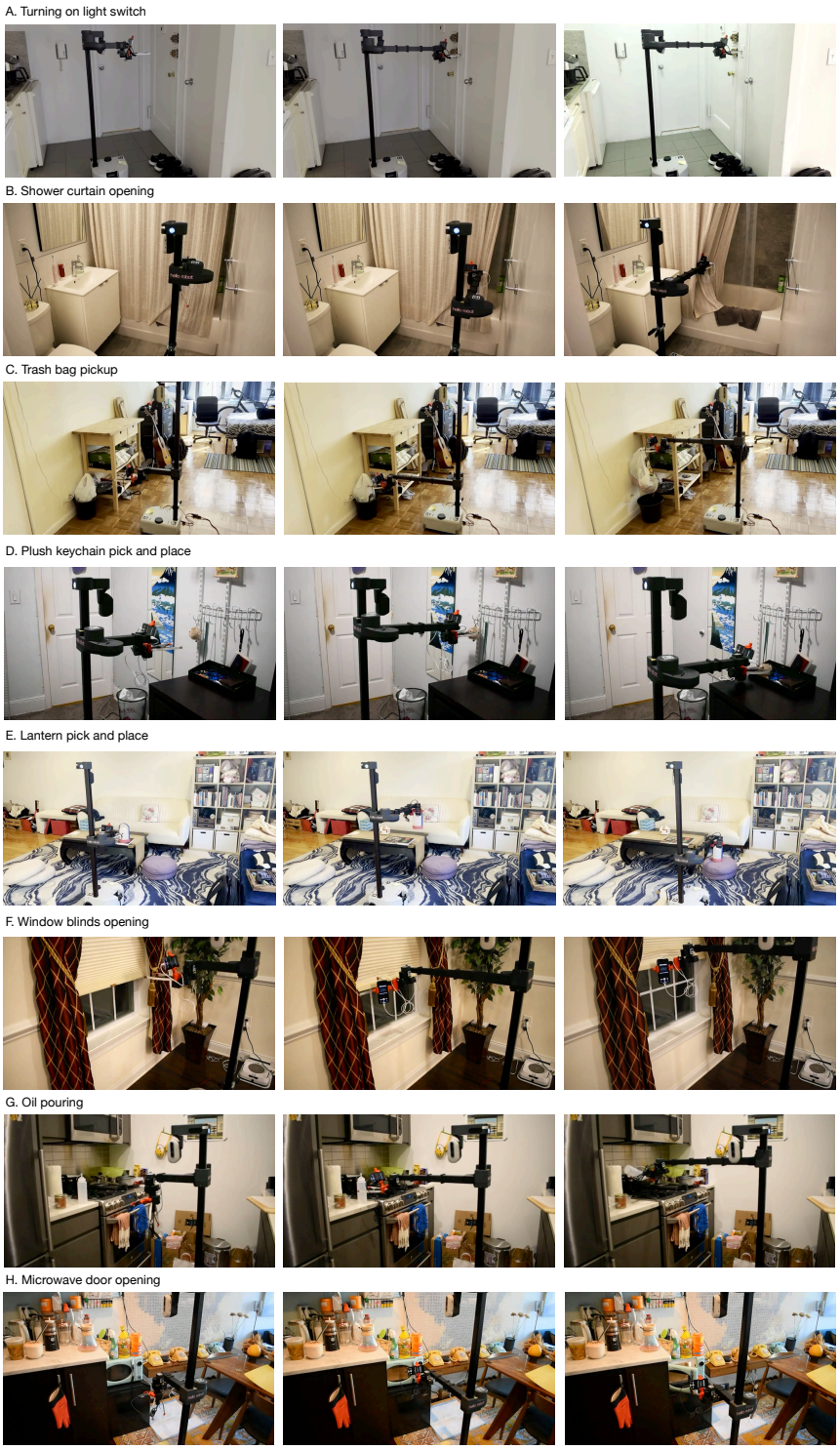


Figure 6.10: A small subset of 8 robot rollouts from the 109 tasks that we tried in homes. A complete set of rollout videos can also be found at our website: https://dobb-e.com/#videos

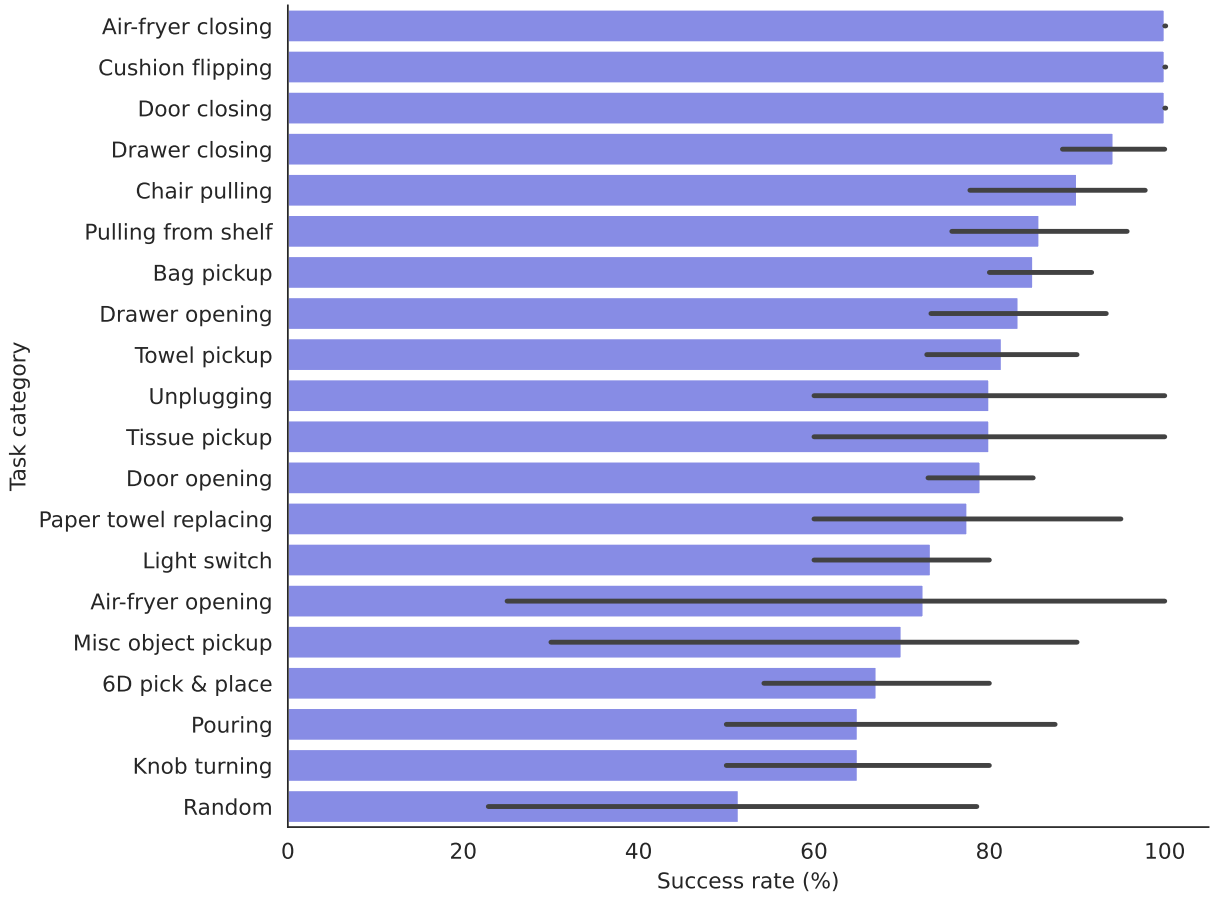


Figure 6.11: Success rate of our 20 different task groups, with the variance in each group's success rate shown in the error bar.

6.3.2 Understanding the Performance of Dobb·E

On a broad level, we cluster our tasks into 20 broad categories, 19 task specific and one for the miscellaneous tasks. There are clear patterns in how easy or difficult different tasks may be, compared to each other.

6.3.2.1 Breakdown by Task Type

We can see from Figure 6.11 that Air Fryer Closing and Cushion Flipping are the task groups with the highest average success rate (100%) while the task group with the lowest success rate is 6D pick

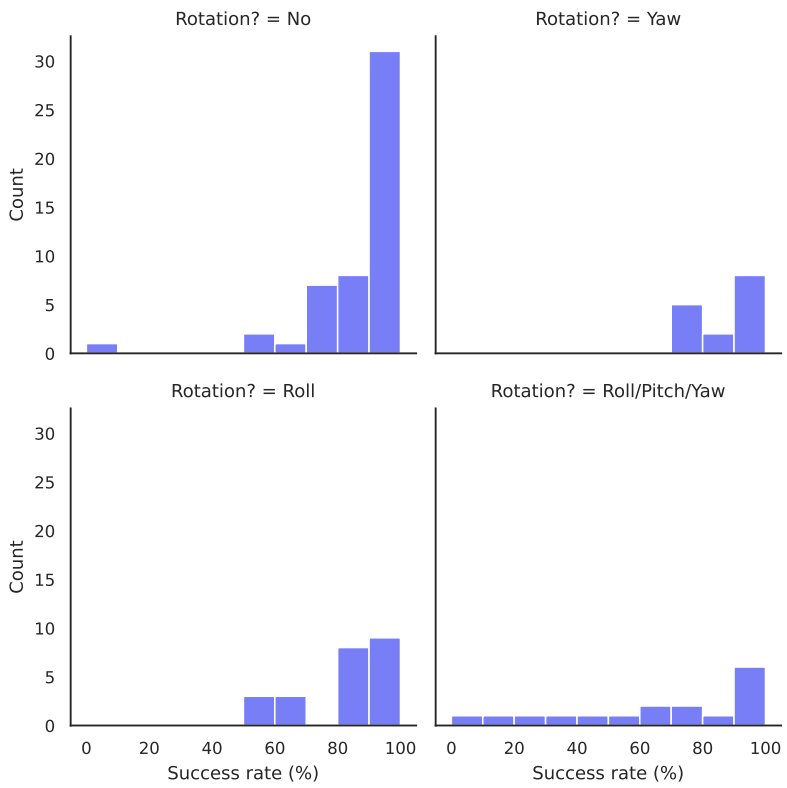


Figure 6.12: Success rate breakdown by type of actions needed to solve the task. The X-axis shows the number of successes out of 10 rollouts, and the Y-axis shows number of tasks with the corresponding number of success.

& place (56%). We found that 6D pick and place tasks generally fail because they generally require robot motion in a variety of axes: like translations and rotations at different axes at different parts of the trajectory, and we believe more data may alleviate the issue. We discuss the failure cases further in Section 6.3.3.

6.3.2.2 Breakdown by Action Type

We can cluster the tasks into buckets by their difficulty as shown in Figure 6.12. We find that the type of movement affects the success rate of the tasks. Specifically, the distribution of success rates for tasks which do not require any wrist rotation is skewed much more positively compared to tasks where we need either yaw or roll, or a combination of yaw, pitch, and roll. Moreover,

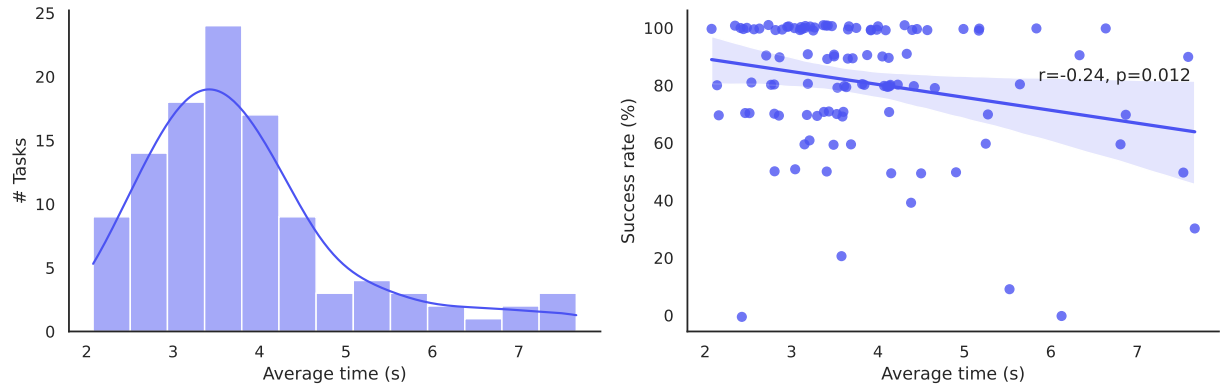


Figure 6.13: (a) Distribution of time (in seconds) taken to demonstrate a task on our experiment setup. The mean time taken to complete one demonstration is 3.82 seconds, and the median time taken is 3.49 seconds. (b) Correlation analysis between time taken to demonstrate a task and the success rate of the associated robot policy.

the distribution of successes for tasks which require 6D motion is the flattest, which shows that tasks requiring full 6D motions are harder compared to tasks where Dobb·E doesn't require full 6D motion.

6.3.2.3 Correlation between demo time and difficulty

Here, we try to analyze the relationship between the difficulty of a task group when done by the robot, and the time required to complete the task by a human. To understand the relationship between these two variables related to a task, we perform a regression analysis between them.

We see from Figure 6.13 that there is a weak negative correlation ($r = -0.24$, with $p = 0.012 < 0.05$) between the amount of time taken to complete a demo by the human demonstrator and how successful the robot is at completing the task. This analysis implies that while longer tasks may be harder for the robot to accomplish, there are other factors that contribute to making a task easy or difficult.

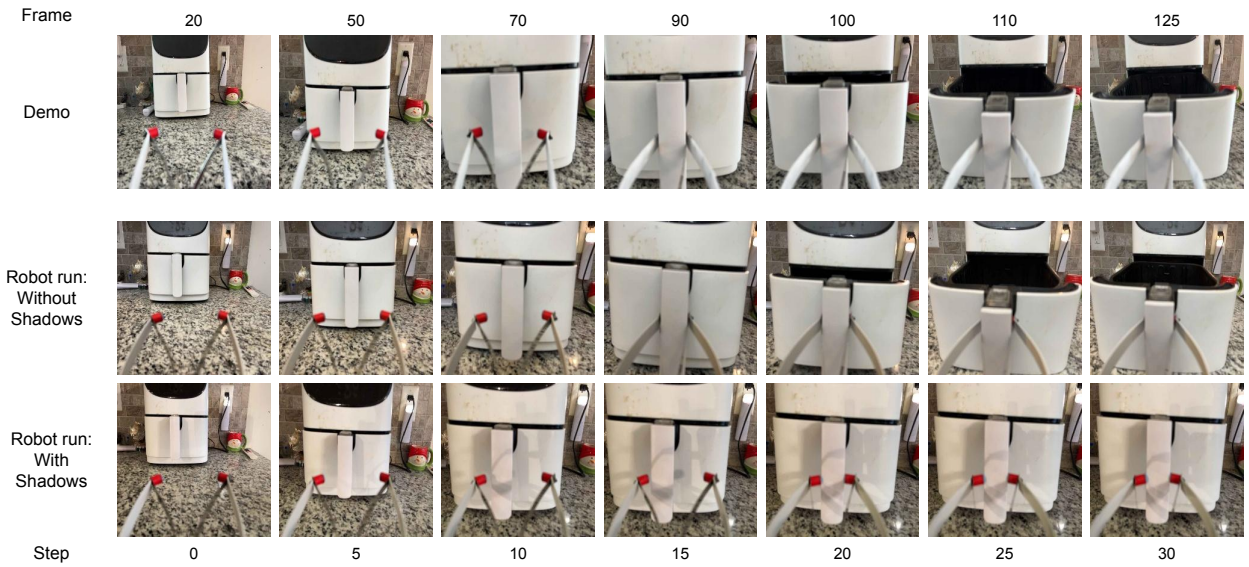


Figure 6.14: First-person POV rollouts of Home 1 Air Fryer Opening comparing (top row) the original demonstration environment, against robot performance in environments with (middle row) similar lighting, and (bottom row) altered lighting conditions with additional shadows.

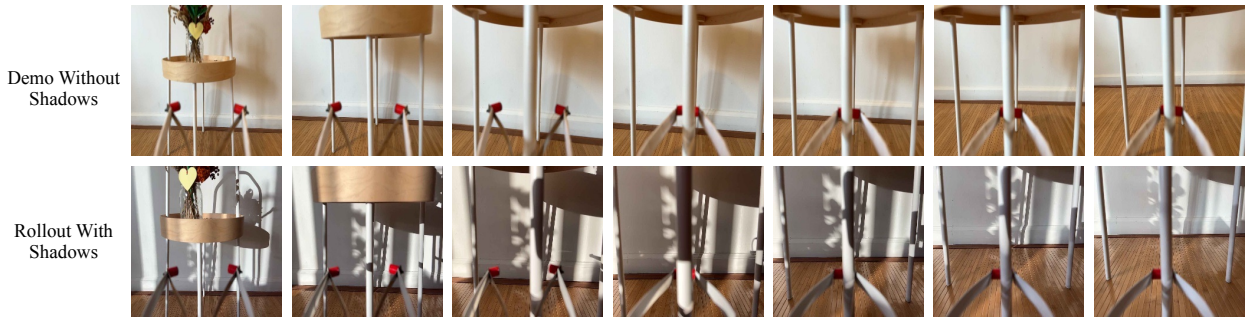


Figure 6.15: First person view from the iPhone from the (top row) Stick during demonstration collection and (bottom row) the robot camera during rollout. Even with strong shadows during rollout, the policy succeeds in pulling the table.

6.3.3 Failure Modes and Analysis

6.3.3.1 Lighting and shadows

In many cases, the demos were collected in different lighting conditions than the policy execution. Generally, with enough ambient lighting, our policies succeeded regardless of day and night conditions. However, we found that if there was a strong shadow across the task space during

Figure 6.16: First person view from the iPhone from the (top row) Stick during demo collection and (bottom row) robot camera during rollout. The demonstrations were collected during early afternoon while rollouts happened at night; but because of the iPhone's low light photography capabilities, the robot view is similar.

execution that was not there during data collection, the policy may behave erratically.

The primary example of this is from Home 1 Air Fryer Opening (see Figure 6.14), where the strong shadow of the robot arm caused our policy to fail. Once we turned on an overhead light for even lighting, there were no more failures. However, this shadow issue is not consistent, as we can see in Figure 6.15, where the robot performs the Home 6 table pulling task successfully despite strong shadows.

In many cases with lighting variations, the low-light photography capabilities of the iPhone helped us generalize across lighting conditions. For example, in Home 8 cushion straightening (Figure 6.16), we collected demos during the day and ran the robot during the night. However, from the robot perspective the difference in light levels is negligible.

6.3.3.2 Sensor limitations

One of the limitations of our system is that we use a lidar-based depth sensor on the iPhone. Lidar systems are generally brittle at detecting and capturing the depth of shiny and reflective objects. As a result, around reflective surfaces we may get a lot of out-of-distribution values on our depth channel and our policies can struggle.

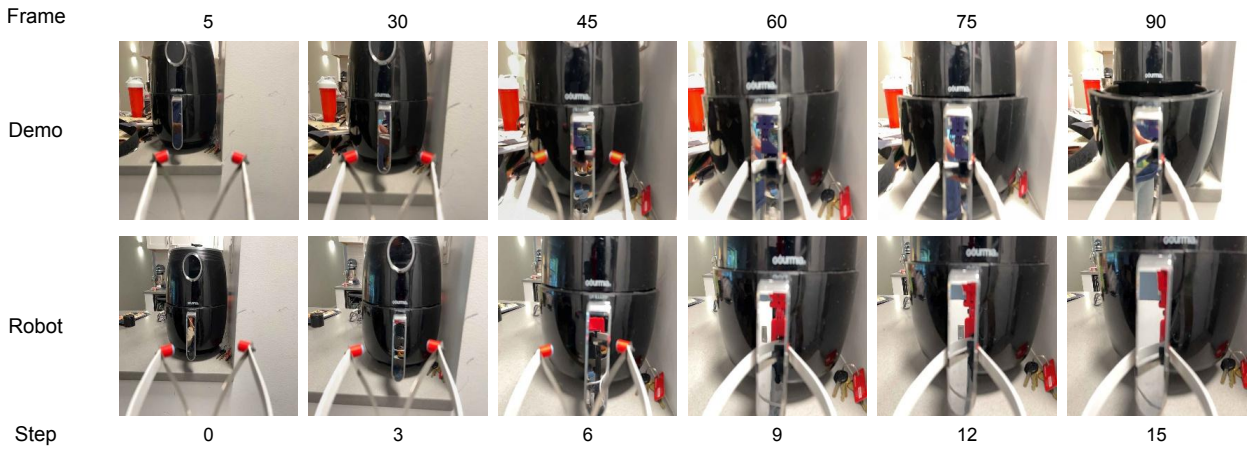


Figure 6.17: First-person POV rollouts of Home 3 Air Fryer Opening showcasing (top row) a demonstration of the task and (bottom row) robot execution.

A secondary problem with reflective surfaces like mirrors is that we collect demonstrations using the Stick but run the trained policies on the robot. In front of a mirror, the demonstration may actually end up recording the demo collector in the mirror. Then, once the policy is executed on the robot, the reflection on the mirror captures the robot instead of the demonstrator, and so the policy goes out-of-distribution and fails.

One of the primary examples of this is Home 3 Air Fryer Opening (Figure 6.17). There, the air fryer handle was shiny, and so had both bad depth and captured the demonstration collector reflection which was different from the robot reflection. As a result, we had 0/10 successes on this task.

Another example is Home 1 vertical window blinds opening, where the camera faced outwards in the dark and provided many out-of-distribution values for the depth (Figure 6.18). In this task, depth-free models performed better (10/10 successes) than depth-using models (2/10 successes) because of such values.

6.3.3.3 Robot hardware limitations

Our robot platform, Hello Robot Stretch RE1, was robust enough that we were able to run all the home experiments on a single robot with only minor repairs. However, there are certain hardware

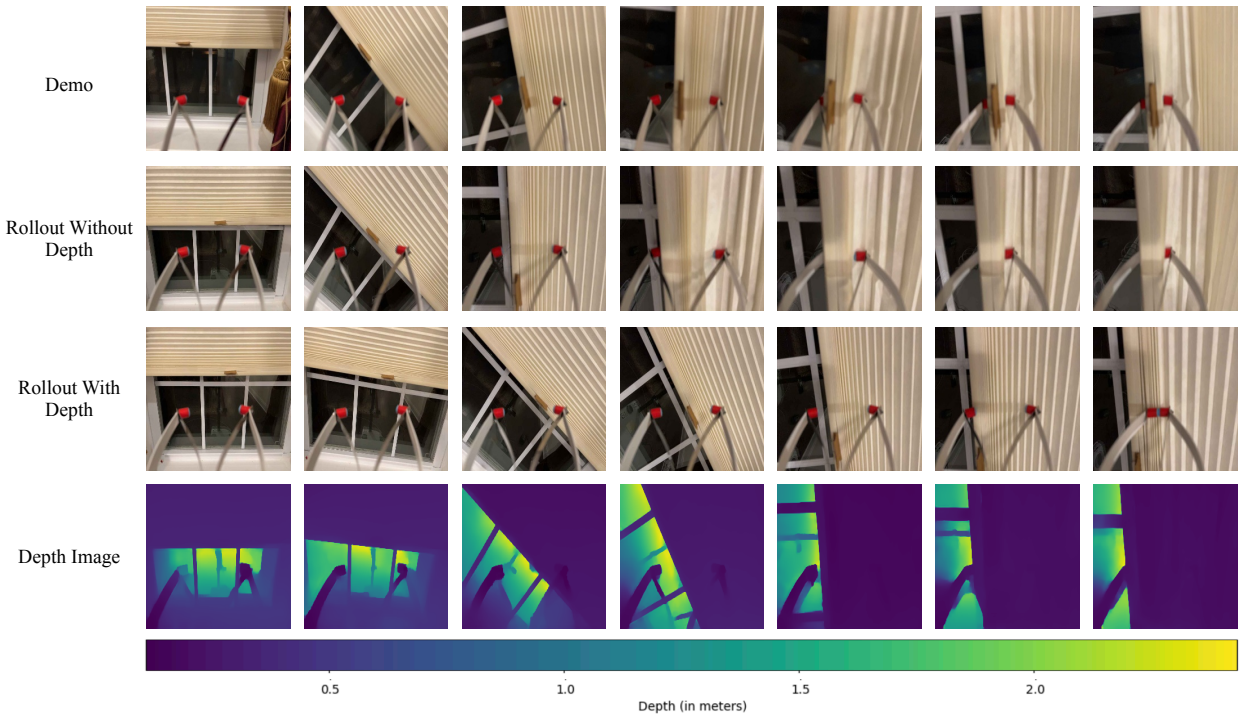


Figure 6.18: Opening an outward facing window blind (top row) both without depth (second row) and with depth (third row). The depth values (bottom row) for objects outside the window are high noisy, which cause the depth-aware behavior model to go out of distribution.

limitations that caused several of our tasks to fail.

The primary constraint we faced was the robot's height limit. While the Stretch is tall, the manipulation space caps out at 1m, and thus a lot of tasks like light switch flicking or picking and placing from a high position are hard for the robot to do. Another challenge with the robot is that since the robot is tall and bottom-heavy, putting a lot of pulling or pushing force with the arm near the top of the robot would tilt the robot rather than moving the arm (Figure 6.19), which was discussed in [Kemp et al. 2022]. Comparatively, the robot was much more successful at opening heavy doors and pulling heavy objects when they were closer to the ground than not, as shown in the same figure. A study of such comparative pulling forces needed can be found in [Jain et al. 2010; Jain and Kemp 2013].

Knob turning, another low performing task, had 65% success rate because of the fine manipulation

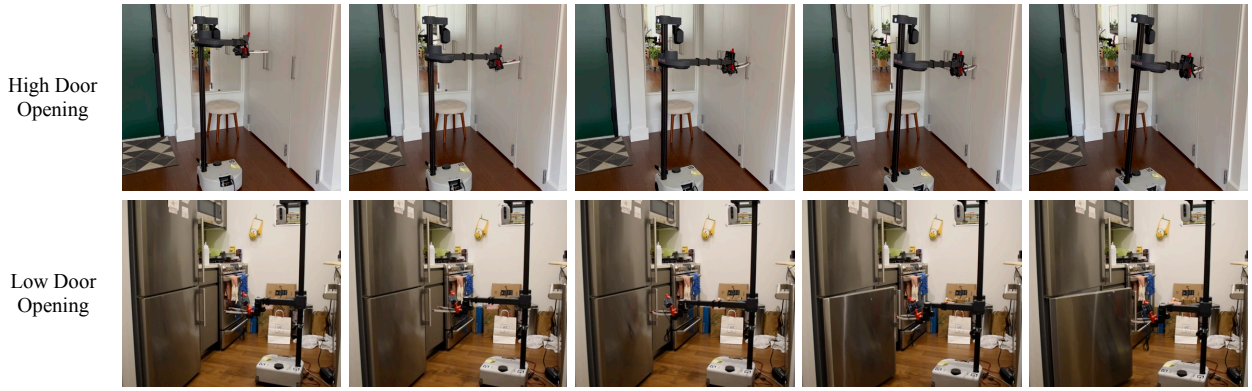


Figure 6.19: The robot pulling on a heavy door handle (top row) high up from the ground and (bottom row) closer the ground. Since the robot is bottom heavy, the first case starts tipping the robot while the second case succeeds.

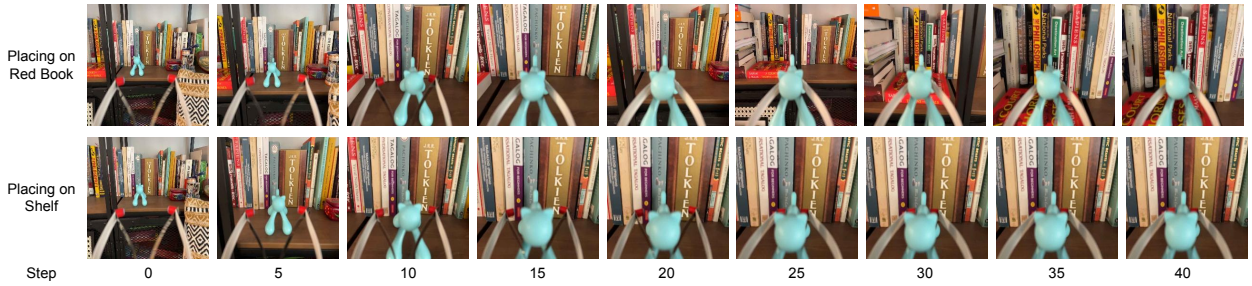


Figure 6.20: First-person POV rollouts of Home 3 Pick and Place comparing (top) a policy trained on demos where the object is picked and placed onto a red book on a different shelf and (bottom) a policy trained on demos where the object is picked and placed onto that same shelf without a red book. In the second case, since there is no clear signal for when to place the object, the BC policy keeps moving left and fails to complete the task.

required: if the robot's grasp is not perfectly centered on the knob, the robot may easily move the wrist without moving the knob properly.

6.3.3.4 Temporal dependencies

Finally, while our policy only relies on the last observations, for a lot of tasks, being able to consider temporal dependency would give us a much more capable policy class. For example, for a lot of Pick and Place tasks, the camera view right after picking up an object and the view right before placing the object may look the same. In that case, a policy that is not aware of time

or previous observations gets confused and can't decide between moving forward and moving backwards. A clear example of this is in Home 3 Pick and Place onto shelf (Figure 6.20), where the policy is not able to place the object if the pick location and the place location (two shelf racks) look exactly the same, resulting in 0/10 successes. However, if the policy is trained to pick and place the exact same object on a different surface (here, a red book on the shelf rack), the model succeeds 7/10 times. A policy with temporal knowledge [Brohan et al. 2023a; Chi et al. 2023; Shafiullah et al. 2022] could solve this issue.

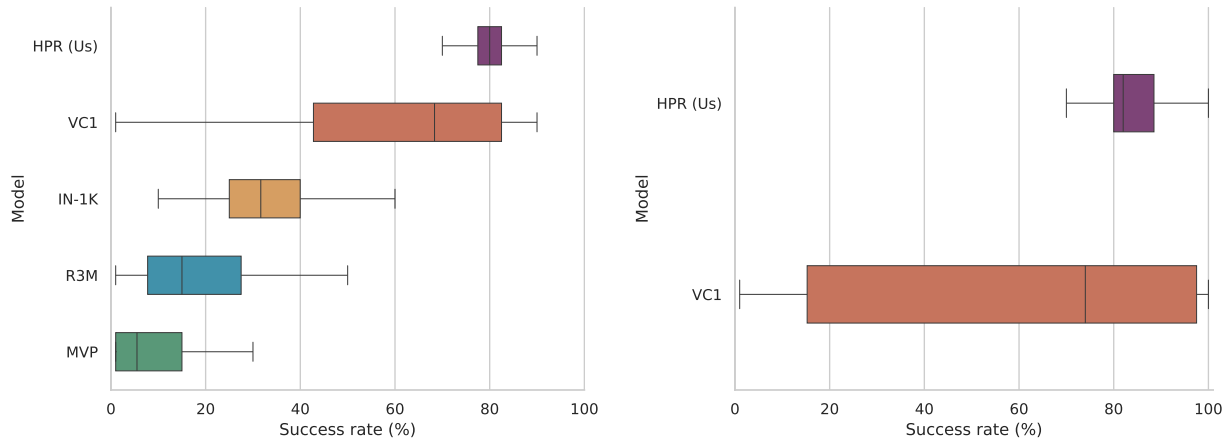
6.3.4 Ablations

We created a benchmark set of tasks in our lab, with a setup that closely resembles a home, to be able to easily run a set of ablation experiments for our framework. To compare various parts of our system, we compare them with alternate choices, and show the relative performance in different tasks. These ablation experiments evaluate different components of our system and how they contribute to our performance. The primary elements of our model that we ran ablations over are the visual representation, number of demonstrations required for our tasks, depth perception, expertise of the demonstrator, and the need for a parametric policy.

6.3.4.1 Alternate visual representation models

Our alternate visual representation comparison is with other pretrained representation models such as MVP [Xiao et al. 2022], R3M [Nair et al. 2022b], VC1 [Majumdar et al. 2023], and a pretrained ImageNet-1k [He et al. 2016; Deng et al. 2009] model. We compare them against our own pretrained models on the benchmark tasks, and compare the performances.

We see that in our benchmark environments, VC1 is the only representation that comes close to our trained representation. As a result, we ran some more experiments with VC1 representation in a household environment. As we can see, while VC1 is closer in performance to our model



(a) Lab tasks **(b)** Home tasks

Figure 6.21: Comparison between different representation models at a set of tasks done in (a) our lab and (b) in a real home enviroment. As we can see, VC-1 is the representation model closest to ours in performance, however it has a high variance behavior where it either performs well or fails to complete the task entirely. The X-axis shows task completion rate distribution with the error bars showing the 95% confidence interval.

compared to IN-1K, R3M and MVP, it under-performs our model in household environments. However, VC-1 shows an interesting pattern of bimodal behavior: in each enviroment it either performs comparatively to HPR, or fails to complete the task entirely.

6.3.4.2 Number of demonstrations required for tasks

While we perform all our tasks with 24 demonstrations each, different tasks may require different numbers of demonstrations. In this set of experiments, we show how models trained on different numbers of demonstrations compare to each other.

As we see in Figure 6.22, adding more demonstrations always improves the performance of our system. Moreover, we see that the performance of the model scales with the number of demonstrations until it saturates. This shows us that on the average case, if our model can somewhat solve a task, we can improve the performance of the system by simply adding more demonstrations.

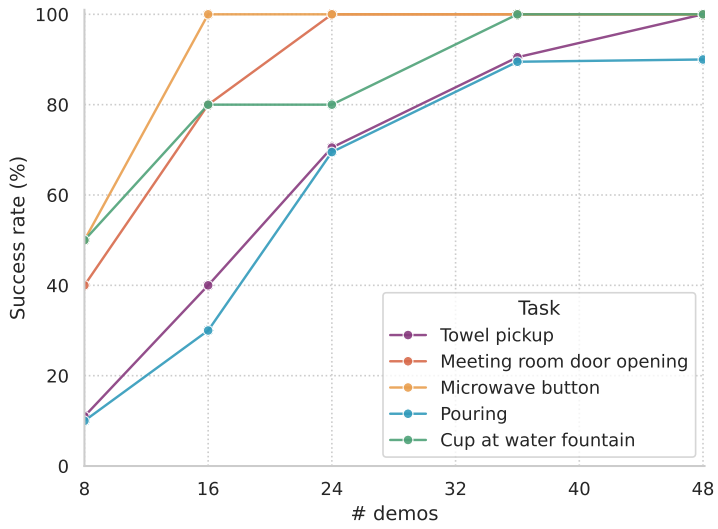


Figure 6.22: Success rates for a given number of demonstrations for five different tasks. We see how the success rate converges as the number of demonstrations increase.

6.3.4.3 Depth Perception

In this work, we use depth information from the iPhone to give our model approximate knowledge of the 3D structure of the world. Comparing the models trained with and without depth in Figure 6.23, we can see that adding depth perception to the model helps it perform much better than the model with RGB-only input.

The failure modes for tasks without depth are generally concentrated around cases where the robot end-effector (and thus the camera) is very close to some featureless task object, for example a door or a drawer. Because such scenes do not have many features, it is hard for a purely visual imitation model without any depth information to know when exactly to close the gripper. On the other hand, the depth model can judge by the distance between the camera and the task surface when to open or close the gripper.

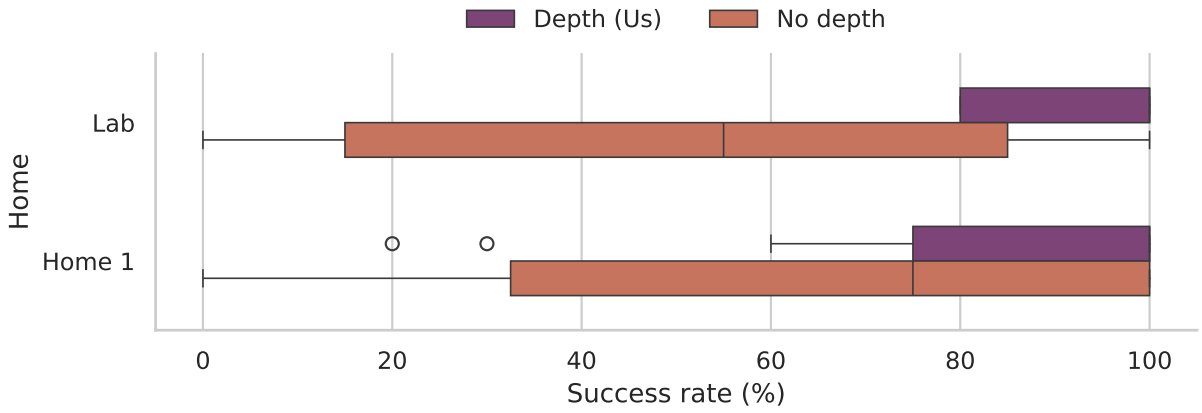


Figure 6.23: Barplot showing the distribution of task success rates in our two setups, one using depth and another not using depth. In most settings, using depth outperforms not using depth. However, there are some exceptional cases which are discussed in Section 6.3.3.2.

6.3.4.4 Demonstrator Expertise

Over the course of our project, we gained experience of how to collect demonstrations with the Stick. A question still remains of how much expertise is needed to operate the Stick and collect workable demonstrations with it.

For this experiment, we have two novice demonstrators collect demonstrations for two tasks in our lab setup. In Task 1, our collected data gave 100% success, while in Task 2, our collected data gave 70% success. Novice collector 1 collected data for Task 1 first and Task 2 second, while collector 2 collected data for Task 2 first and Task 1 second. Collector 1's data had 10% success rate on Task 1, but had 70% success on Task 2. Collector 2's data had 0% success on Task 2 but 90% success on Task 1. From the data, we can see that while it may not be trivial initially to collect demonstrations and teach the robot new skills, with some practice both of our demonstrators were able to collect demonstrations that were sufficient.

Figure 6.24: Open-loop rollouts from our demonstrations where the robot actions were extracted using (a) the odometry from iPhone and (b) OpenSfM respectively.

6.3.4.5 Odometry

In our system, we used the Stick odometry information based on the iPhone's odometry estimate. Previous demonstration collection systems in works like [Young et al. 2021; Pari et al. 2021] used structure-from-motion based visual odometry methods instead, like COLMAP [Schonberger and Frahm 2016] and OpenSfM [Adorjan 2016]. In this section, we show the difference between the iPhone's hardware-based and OpenSfM's visual odometry methods, and compare the quality of the actions extracted from them.

As we can see from the Figure 6.24, OpenSfM-extracted actions are generally okay while the camera is far away from everything. However, it fails as soon as the camera gets very close to any surface and loses all visual features. The hardware odometry from the iPhone is much more robust, and thus the actions extracted from it are also reliable regardless of the camera view.

6.4 Open Problems and Future Research

In this work we have presented an approach to scalable imitation learning that can be applied in household settings. However, there remains open problems that we must address before truly being able to bring robots to homes.

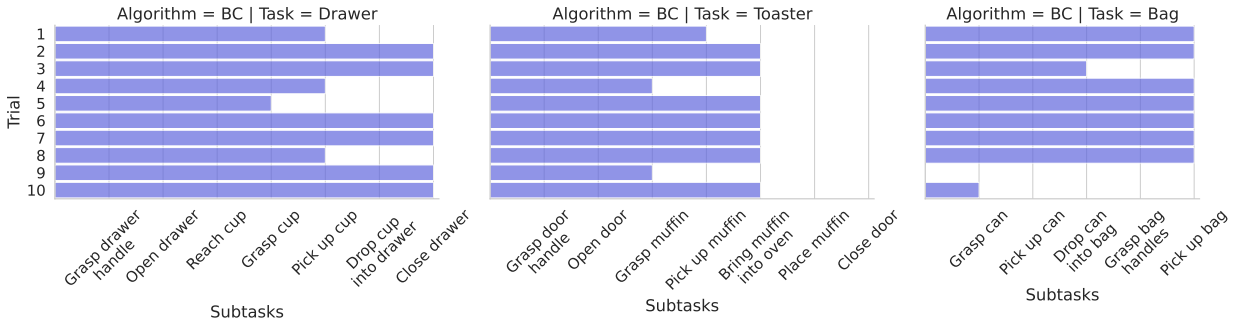
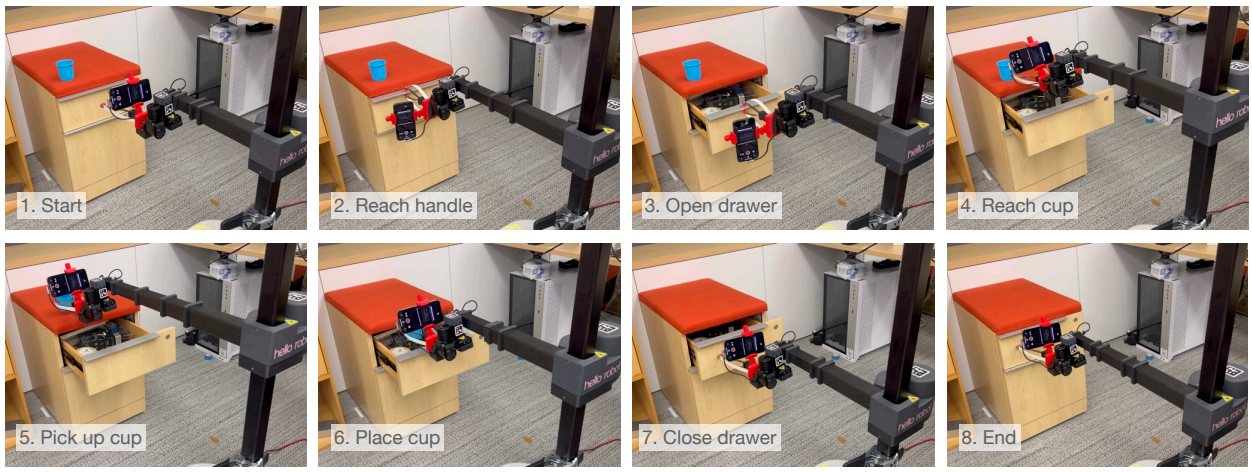


Figure 6.25: Analysis of our long-horizon tasks by subtasks. We see that Dobb·E can chain subtasks, although the errors can accumulate and make overall task success rate low.

6.4.1 Scaling to Long Horizon Tasks

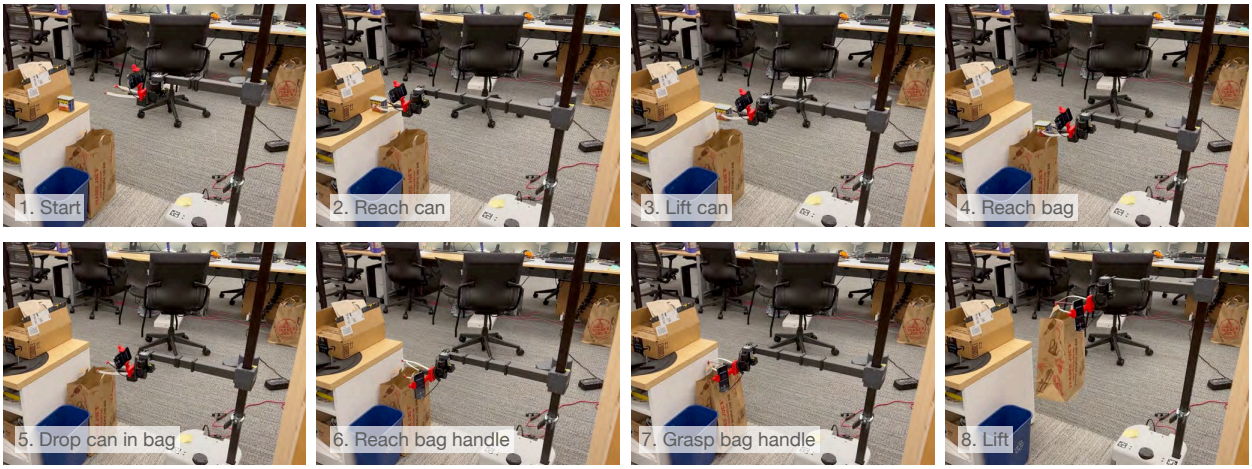
We primarily focused on short-horizon tasks in this work, but intuitively, our framework should be easily extensible to longer-horizon, multi-step tasks with algorithmic improvements. To validate this intuition, we train Dobb·E to perform some multi-step tasks in our lab.

In Figures 6.26(a), 6.26(b), and 6.26(c), we can see that Dobb·E can successfully perform multi-step, long horizon tasks like putting a cup in a drawer, placing a muffin in a toaster oven, or placing a can in a recycling bag and lifting it. However, because of the compound nature of these tasks, the failure cases also tend to compound with our simple methods, as seen in Figure 6.25. For example, in the muffin-in-toaster task, our model got 1 success out of 10 trials, and in the cup-in-drawer task, our model got 6 success out of 10 trials. In both cases, the sub-task causing primary failure was not letting go of the grasped object (cup or muffin). If we can improve on such particular subtasks, possibly using force-aware methods similar to [Collins et al. 2023b], we believe Dobb·E can easily scale up to long-horizon tasks. Fast on-line adaptation on top of offline training [Haldar et al. 2023a,b] has potential to improve such long horizon cases as well. In other cases, the robot was able to open the door but unable to disengage safely from the handle because some part of the robot gripper got stuck to the handle. This failure mode points to the need of better designed, less bare-boned robot grippers for household tasks.



(a) The robot opening a drawer, placing a cup inside of it, and closing it afterwards.

(b) The robot opening a toaster oven, placing a muffin inside of it, and closing it.



(c) The robot picking up a can, placing it in a bag, and then lifting it.

Figure 6.26: Dobb·E completing three temporally extended tasks each made up of five to seven subtasks.

6.4.2 Incorporating Memory

Another large challenge in our setup is the problem of robotic scene memory. With a single first person point of view on the Stick, the robot needs to either see or remember large parts of the scene to operate on it effectively. However, there is a dearth of algorithms that can act as standalone memory module for robots. The algorithms that currently exist, such as [Shafiullah et al. 2023a; Kerr et al. 2023; Rashid et al. 2023; Wang et al. 2023b; Shen et al. 2023; Jatavallabhula et al. 2023; Bolte et al. 2023; Huang et al. 2023b] also tend to have a rigid representation of the scene that is hard to change or edit on the fly, which will need to improve for real household deployments.

6.4.3 Improving Sensors and Sensory Representations

Most of current visual representation learning algorithms focus on learning from third-person views, since that is the dominant framework in Computer Vision. However, third person cameras often rely on camera calibration, which generally makes using large robot datasets and transferring data between robots difficult [Bharadhwaj et al. 2023]. A closer focus on learning from first person cameras and eye-in-hand cameras would make sharing data from different environments, tasks, and robots much easier. Finally, one of the modality that our Stick is missing is having tactile and force sensors on the gripper. In deployment, we have observed the robot sometimes applies too much or too little force because our framework doesn't contain such sensors. Better integration of cheap sensors [Bhirangi et al. 2021] with simple data collection tools like the Stick, or even more methods like learned visual contact force estimation [Grady et al. 2022; Collins et al. 2023a] could be crucial in such settings.

6.4.4 Robustifying Robot Hardware

A large limitation on any home robotics project is the availability of cheap and versatile robot platforms. While we are able to teach the Hello Robot Stretch a wide-variety of tasks, there were many more tasks that we could not attempt given the physical limitations of the robot: its height, maximum force output, or dexterous capabilities. Some of these tasks may be possible while teleoperating the robot directly rather than using the Stick, since the demonstrator can be creative and work around the limits. However, availability of various home-ready robotic platforms and further development of such demonstration tools would go a long way to accelerate the creation of household robot algorithms and frameworks.

6.5 Reproducibility and Call for Collaboration

To make progress in home robotics it is essential for research projects to contribute back to the pool of shared knowledge. To this end, we have open-sourced practically every piece of this project, including hardware designs, code, dataset, and models. Our primary source of documentation for getting started with Dobb·E can be found at https://docs.dobb-e.com.

- **Robot base:** Our project uses Hello Robot Stretch as a platform, which is similarly open sourced and commercially available on the market for US$24,000 as of November 2023.
- **Hardware design:** We have shared our 3D-printable STL files for the gripper and robot attachment in the GitHub repo: https://github.com/notmahi/dobb-e/tree/main/hardware. We have also created some tutorial videos on putting the pieces together and shared them on our website. The reacher-grabber stick can be bought at online retailers, links to which are also shared on our website https://dobb-e.com/#hardware.
- **Dataset:** Our collected home dataset is shared on our website. We share two versions, a 814 MB

version with the RGB videos and the actions, and an 77 GB version with RGB, depth, and the actions. They can be downloaded from our website, https://dobb-e.com/#dataset. At the same time, we share our dataset preprocessing code in GitHub https://github.com/notmahi/dobb-e/tree/main/stick-data-collection so that anyone can export their collected R3D files to the same format.

- **Pretrained model:** We have shared our visual pretraining code as well as checkpoints of our pretrained visual model in our GitHub https://github.com/notmahi/dobb-e/tree/main/imitation-in-homes and Huggingface Hub https://huggingface.co/notmahi/dobb-e. For this work, we also created a high efficiency video dataloader for robotic workload, which is also shared under the same GitHub repository.
- **Robot deployment:** We have shared our pretrained model fine-tuning code in https://github.com/notmahi/dobb-e/tree/main/imitation-in-homes, and the robot controller code in https://github.com/notmahi/dobb-e/tree/main/robot-server. We also shared a step-by-step guide to deploying this system in a household, as well as best practices that we found during our experiments, in a handbook under https://docs.dobb-e.com.

Beyond these shared resources, we are also happy to help other researchers set up this framework in their own labs or homes. We have set up a form on our website to schedule 30-minutes online meetings, and shared some available calendar slots where we would be available to meet online and help set up this system. We hoping these steps would be beneficial for practitioners to quickly get started with our framework.

Finally, we believe that our work is an early step towards learned household robots, and thus can be improved in many possible ways. So, we welcome contributions to our repositories and our datasets, and invite researchers to contact us with their contributions. We would be happy to share such contributions with the world with proper credits given to the contributors.

Postscript

Dobb·E is a depth-first inquiry into the application of learning-based robotics into homes, finding ways in which robots are able to solve problems in real world settings. On the way to this achievement, we were able to create innovative data collection systems, diverse datasets (more in Chapter 7), lightweight learning algorithms, and an large array of tasks that are suddenly within bounds for robots in wild environments.

This work could have been significantly more impressive by focusing on more complex, long-horizon, and dexterous tasks – although that would make the work a lot more resource intensive. Another possible shortcoming of this work is simply trying to do too much – we introduce new hardware for data collection and deployment, new learning algorithms, and a series of diverse empirical learning. As such, contributions end up overshadowing each other. Finally, the system is complete – but it focuses too much on a singular robot embodiment which makes it inaccessible to those without the same robot.

Acknowledgements

This work was co-led with Anant Rai, co-authored with Haritheja Etukuru, Yiqian Liu, Ishan Misra, Soumith Chintala, and advised by Lerrel Pinto. NYU authors are supported by grants from Amazon, Honda, and ONR award numbers N00014-21-1-2404 and N00014-21-1-2758. NMS is supported by the Apple Scholar in AI/ML Fellowship. LP is supported by the Packard Fellowship. Our utmost gratitude goes to our friends and colleagues who helped us by hosting our experiments in their homes, and those who helped us collect the pretraining data. We thank Binit Shah and Blaine Matulevich for support on the Hello Robot Platform and the NYU HPC team, especially Shenglong Wang, for compute support. We thank Jyo Pari and Anya Zorin for their work on

earlier iterations of the Stick. We additionally thank Sandeep Menon and Steve Hai for his help in the early stages of data collection. We thank Paula Nina and Alexa Gross for their input on the designs and visuals. We thank Chris Paxton, Ken Goldberg, Aaron Edsinger, and Charlie Kemp for feedback on early versions of this work. Finally, we thank Zichen Jeff Cui, Siddhant Haldar, Ulyana Pieterberg, Ben Evans, and Darcy Tang for the valuable conversations that pushed this work forward.

7 General Policies for Zero-Shot Deployment in New Environments: Robot Utility Models

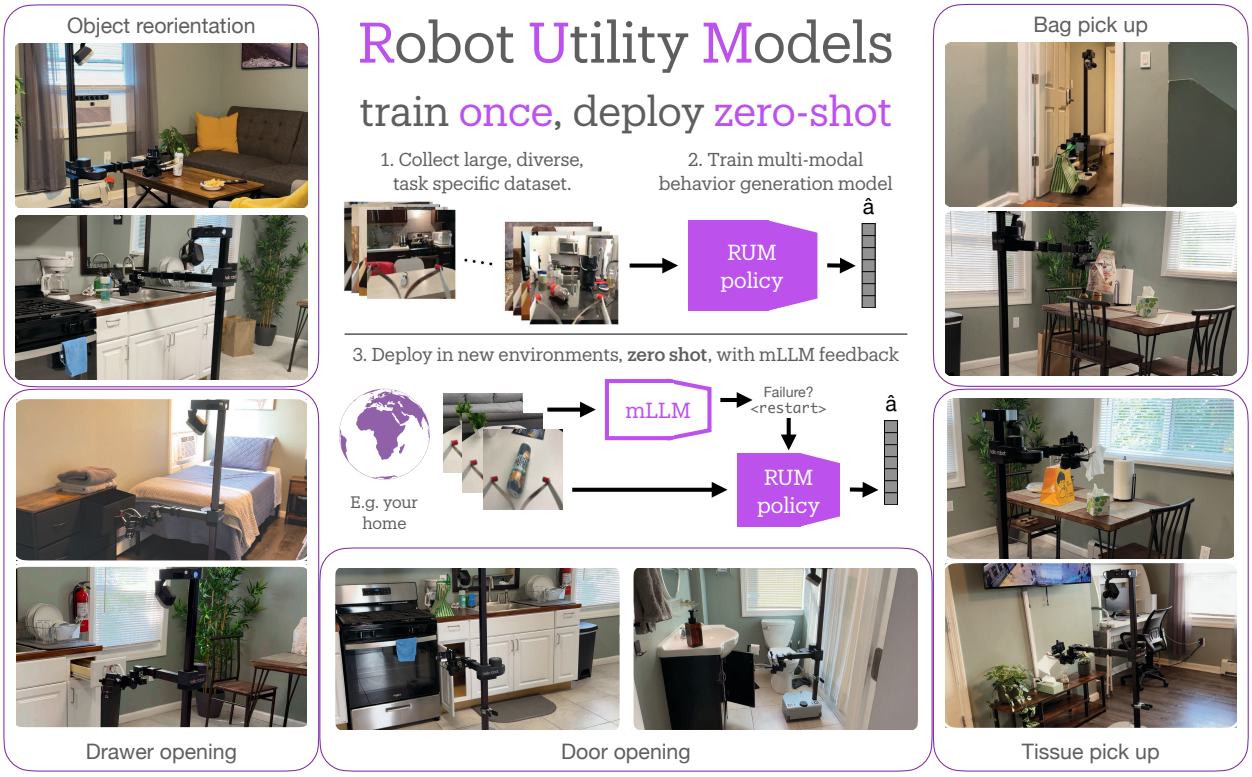


Figure 7.1: Robot Utility Models are trained on a diverse set of environments and objects, and then can be deployed in novel environments with novel objects without any further data or training.

7.1 Introduction

We have seen rapid progress in training manipulation skills recently [Zhao et al. 2023b; Fu et al. 2024b; Zitkovich et al. 2023; Haldar et al. 2024; Fu et al. 2024a; Lin et al. 2024; Kim et al. 2024], largely brought about by fitting deep networks on data collected by teleoperating robots [Mandlekar et al. 2018; Iyer et al. 2024; Arunachalam et al. 2023b; Cheng et al. 2024; Khazatsky et al. 2024]. The mechanism for deploying such skills in new environments mimics the pretrain-then-finetune strategy first developed by the vision community circa 2014 [Girshick et al. 2014]. There, models were first pretrained on ImageNet and then finetuned on task-specific data such as detection, segmentation, and pose estimation [Girshick et al. 2014; Gkioxari et al. 2014]. In the context of robotics, this strategy involves pretraining on large robot datasets [Padalkar et al. 2023; Khazatsky et al. 2024; Shafiullah et al. 2023b; Walke et al. 2023] to produce a robot foundation model, which is then fine-tuned on data collected in new environments or tasks [Shafiullah et al. 2023b; Team et al. 2024; Kim et al. 2024]. This need to fine-tune the foundation model for each and every new environment is limiting as it requires humans to collect data in the very environment where the robot is expected to perform. So while vision and language models have moved on to zero-shot deployments, i.e. without any environment-specific finetuning data, such a capability eludes most robot manipulators. This is not to say that there have not been attempts to create zero-shot manipulation models – several foundational work in grasping and pick-and-place [Fang et al. 2023c; Sundermeyer et al. 2021; Mahler et al. 2017a] have tackled this problem albeit with a task-specific solution.

So what makes creating a general policy for an arbitrary task that can work zero-shot hard? First is the concern about sufficient data – the necessary amount of data to train such a general model could be large. Since collecting robot data is hard, creating a large dataset is also hard and often expensive since humans are usually tasked to collect robot demonstrations. Second, when a large

dataset is collected in the open-world it would necessarily have large diversity and multiple modes in demonstrator behavior. Fitting a robot models on this diverse data is a challenge. Third, unlike vision and language, where the native form of data, i.e. images and text are largely standard, robotics is far from having a standard camera and hardware setup along with physical challenges of running models in realtime on onboard compute. Creating zero-shot models that can run with even minor changes to hardware setup between training and deployment requires careful attention to details. Finally, any model deployed zero shot on a novel environment naturally has a higher failure rate than a model that has been fine-tuned on that environment. Thus, to deploy a model zero-shot, it is important to have a mechanism for error detection and recovery.

In this work, we introduce Robot Utility Models (RUMs), a new framework for training focused and functional *utility* models to complete helpful tasks that can be deployed zero-shot **without further training or fine-tuning** in novel environments. This is done by taking a systems-first approach. To scale up our datasets without compromising on data quality, we develop a new tool, building on prior work in untethered data collection [Shafiullah et al. 2023b; Chi et al. 2024]. We train policies on these diverse dataset with state-of-the-art multi-modal behavior learning algorithms [Lee et al. 2024; Chi et al. 2023] and show how they can absorb and scale with large-scale demonstration data. Finally, we deploy the policy in multiple different environments out of the box, with self-critique via mLLMs [Guo et al. 2023] and retrying, showing how the policy can be robustly executed on cheap, general-purpose hardware. A selection of our trained models are available on the Hello Robot Stretch without much modifications. Beyond the default Stretch deployment, we also enable deployment on other robot arms, cameras, and lighting conditions, showing the generalizability of our approach.

Creating and deploying RUMs led us to several interesting lessons. First, we find that the quantity and quality of data is crucial for training a utility model, with the choice of model architecture being less critical. Second, we see that the diversity of the data collected is crucial for the model

to generalize to new environments, and more important than the raw quantity of data. Third, we find that the model can be made more capable in single environments by performing self-critique on the model performance with an independent model and retrying when appropriate.

To validate RUMs, we run a total of 2,950 robot rollouts in real-world environments including homes in New York City (NY), Jersey City (NJ), and Pittsburgh (PA). These experiments reveal the following:

- We show that it is possible to create general Robot Utility Models with a moderate amount of data in the order of 1,000 demonstrations (Section 7.2). These RUMs achieve a 90% average success rate on zero-shot deployment in 25 novel environments (Section 7.3.1).
- The success of RUMs relies primarily on two key techniques. First, the use of multi-modal policies (Section 7.2.3) provides a zero-shot success rate of 74.4% (Section 7.3.2). Second, the mLLM based self-critique and retrying system (Section 7.2.4) further improves the success rate by 15.6% (Section 7.3.6).
- While the overall framework for RUMs is straightforward, the devil is in the details, where we find gains from unexpected sources, e.g. data diversity vs. data quantity (Section 7.3.4 and 7.3.5).

To encourage the development of RUMs for a wider variety of tasks, our code, data, models, hardware designs, as well as our experiment and deployment videos are open sourced and can be found on our website: robotutilitymodels.com.

7.2 Robot Utility Models

We take a full-stack approach to create Robot Utility Models. At its core, our system follows the imitation learning framework. However, to effectively scale imitation learning to the point where

our trained policies are deployable zero-shot, we create new tools and techniques to improve data collection, model training, inference, and deployment.

7.2.1 Data collection tool

One of the primary requirements of our system is to be able to scale up diverse yet accurate demonstration data for cheap. To this end, we continue on the evolutionary path of hand-held, portable data collection tools [Song et al. 2020; Young et al. 2020; Pari et al. 2021; Shafiullah et al. 2023b; Chi et al. 2024] that let us quickly collect precise demonstrations. Following our previous work [Shafiullah et al. 2023b], we call this tool *Stick-v2*, which is a hand-held data collection tool built out of an iPhone Pro and a bill of materials that adds up to $25. We combine inspirations from the quick deployability of Stick-v1, and the compact, handheld form factor of UMI gripper. For a detailed build instruction and the bill of materials, we refer the reader to the supplementary materials (Appendix E.2.1).

Our design decisions are predicated on a few factors: portability, convenience, and set-up speed. We experimentally found these factors to be important to quickly scale up robot datasets and training RUMs. As we show with experiments in Section 7.3.3, one of the most crucial aspect of data collection for RUMs is data diversity, i.e. collecting data from a large number of diverse environments. Thus, it is crucial to have a portable tool that is easy to mass-print, carry, and deploy in a new environment. Secondly, it is important for the collected data to be accurate across many environments with many variations. Finally, it is important to minimize the "per-environment set-up time", whether that time is spent setting up the data collection system, calibrating the camera, or the tool's SLAM system.

For the above reason, we design our data collection tool, Stick-v2, around the ARKit API from the widely available and used iPhone Pro (Figure 7.2). Given its technical capabilities, the only digital component in our Stick-v2 is this iPhone, which makes our tool particularly robust to

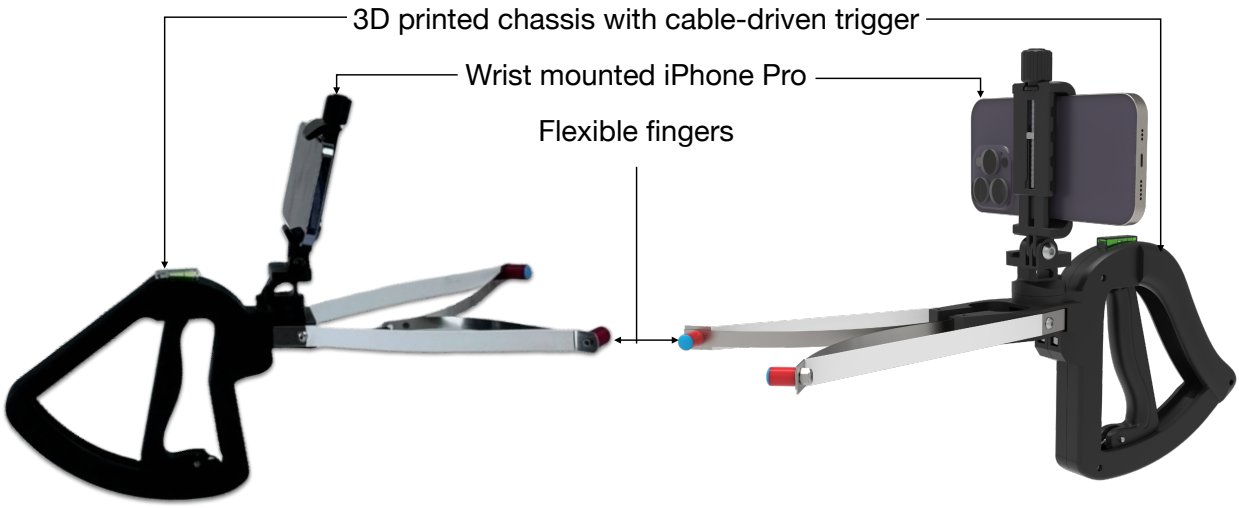


Figure 7.2: Stick-v2, our data collection tool (left: real photo, right: render), is built out of an iPhone Pro and a bill of materials that adds up to $25. The tool is portable, robust, and makes it easy to start collecting data in a new environment in seconds.

shipping and handling. The iPhone, and therefore Stick-v2, can collect RGB video and depth data at up to 60 Hz and high precision 6D pose and position information from the ARKit API at up to 100Hz. To capture the gripper opening information, we trained an RGB-based model that predicts the gripper aperture from images. Furthermore, this data is automatically synchronized and timestamped by the iPhone without the need for any calibration. This allows us to collect data from a wide variety of environments with no set-up time. This is in contrast to other data collection tools based on visual SLAM systems which has limited precision and are non-robust around "textureless" scenes such as close to flat walls, ceilings, or corners [Chi et al. 2024; Young et al. 2020]. Finally, not needing camera calibration makes our system deployable out-of-the-box in any environment, especially in the real world where the environment is not controlled. This enables us to, for example, collect data from retail home goods stores with minimal interruptions to enrich our datasets, which would be hindered if we had to calibrate the camera and odometry system for each new environment.

7.2.2 Collected datasets

We collect data for each of our five tasks, which are as defined below:

- **Door opening:** Open doors with a long handle, on e.g. cabinets and microwaves. Due to hardware limitations, our robot cannot open doors with round knobs, so we exclude them from our dataset.
- **Drawer opening:** Open a drawer with a handle. We exclude drawers with knobs from our dataset for similar reasons as above.
- **Reorientation:** Pick up a cylindrical object (e.g. bottle) lying on a flat surface and place it upright on the same surface.
- **Tissue pickup:** Pick up a soft, flexible tissue paper from any tissue paper box.
- **Bag pickup:** Pick up a kraft paper bag or similar other bags from a flat surface.

For each of our five RUMs, we focused on gathering approximately 1,000 demonstrations on approximately 40 environments, with about 25 demonstrations per environment on average. The only exceptions are door opening with 1,200 and drawer opening with 525 demonstrations. A small collection of such environments are shown in Figure 7.3. For the door opening task, we seeded this dataset with the Homes of New York dataset [Shafiullah et al. 2023b] as well as demonstrations collected during the Dobb·E experiments. For the other tasks, our dataset consists of new demonstrations collected using the Stick-v2 tool on a novel set of environments and objects. For demonstrations collected from the previous dataset by inexperienced data collectors, we do a manual quality check and exclude any environment that has a high number of low-quality demonstrations, such as failed demonstrations. Note that, to keep our experiments unbiased, we hold out test environments and objects and never collect any data on them. To gain quick insight

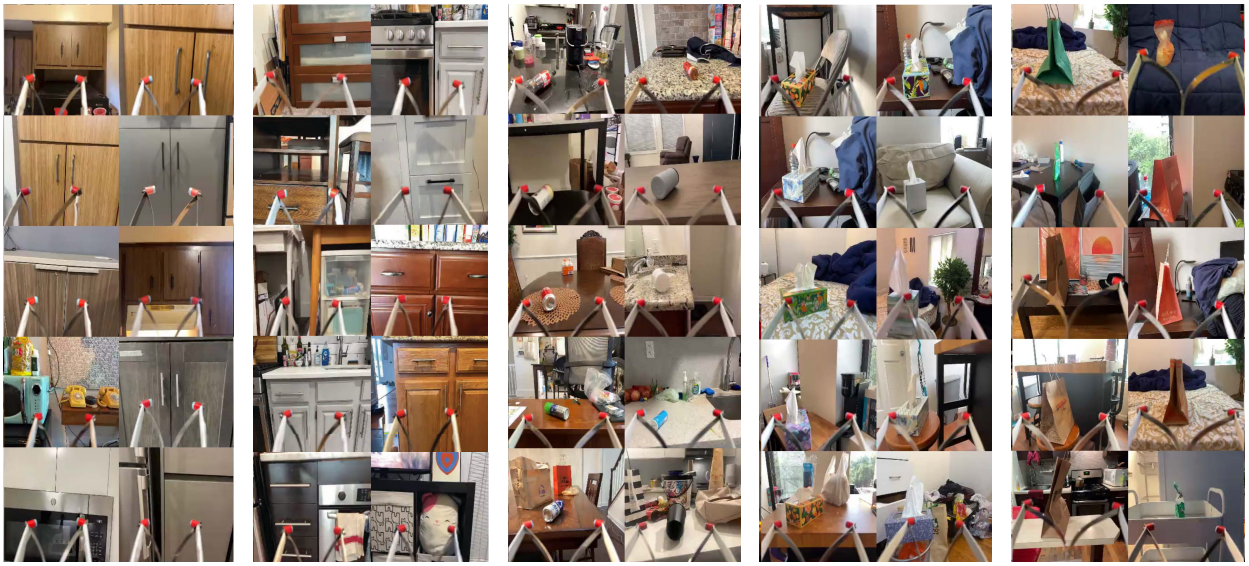


Figure 7.3: A small sample of environment and objects from our collected dataset. We collect data for each of our five tasks on a diverse set of environments and objects using Stick-v2.

on different task data we use for training, we created an interactive data diversity visualization tool: robotutilitymodels.com/data_diversity/.

7.2.3 Model training

Given that our data is collected by a large set of demonstration collectors, conceptually it is important for the model to handle any resultant multi-modality in the dataset. In this work, we train a large set of policy classes on our datasets for each task. Among the policy classes, the best performing ones are VQ-BeT [Lee et al. 2024] and Diffusion Policy (DP) [Chi et al. 2023]. We also train ACT [Zhao et al. 2023b] and MLP-BC policies on a limited set of tasks. Each policy class shares some features, such as a ResNet34-based vision encoder initialized to the HPR encoder from [Shafiullah et al. 2023b], and a transformer-based policy trunk. We also train each model for the same 500 epochs. Beyond that, we sweep to find the best hyperparameters for learning rate, history length, and chunk size, and use the recommended hyperparameters from the original

papers for each model. Our final VQ-BeT models are trained on data subsampled at 3.75Hz, and uses 6 most recent frames of history to predict the next action. All of our models predict the action in relative 6D space for the robot end-effector, and absolute value in the range $[0, 1]$ for the gripper opening. We discuss the impact of choosing different training algorithms in Section 7.3.2. Training all of our models took between 24 and 48 hours on 2 Nvidia A100 GPUs on our cluster, with proportional speed-ups by using more GPUs or using more recent GPUs like H100s.

7.2.4 Retrying with GPT-4o feedback

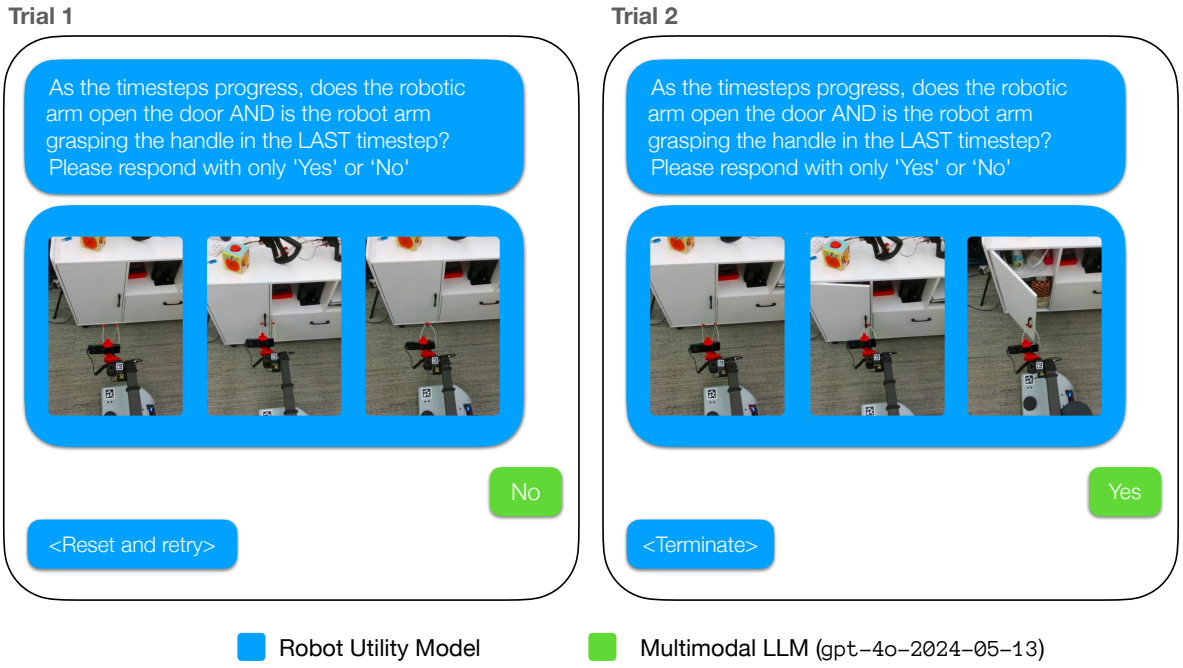


Figure 7.4: Automated retrying with feedback from multimodal LLM critic. We use a multimodal LLM (`gpt-4o-2024-05-13` in our experiments) to verify the success of a task given a summary of robot observations. If the mLLM detects a failure, we automatically reset the robot and retry the task with a new initial robot state until success or timeout.

While a pre-trained model can solve the task in a new environment, to achieve the best possible performance, it is helpful to have additional runtime support for the model. For our deployment, we use an multimodal LLM (`gpt-4o-2024-05-13`) as an introspection module for our policies

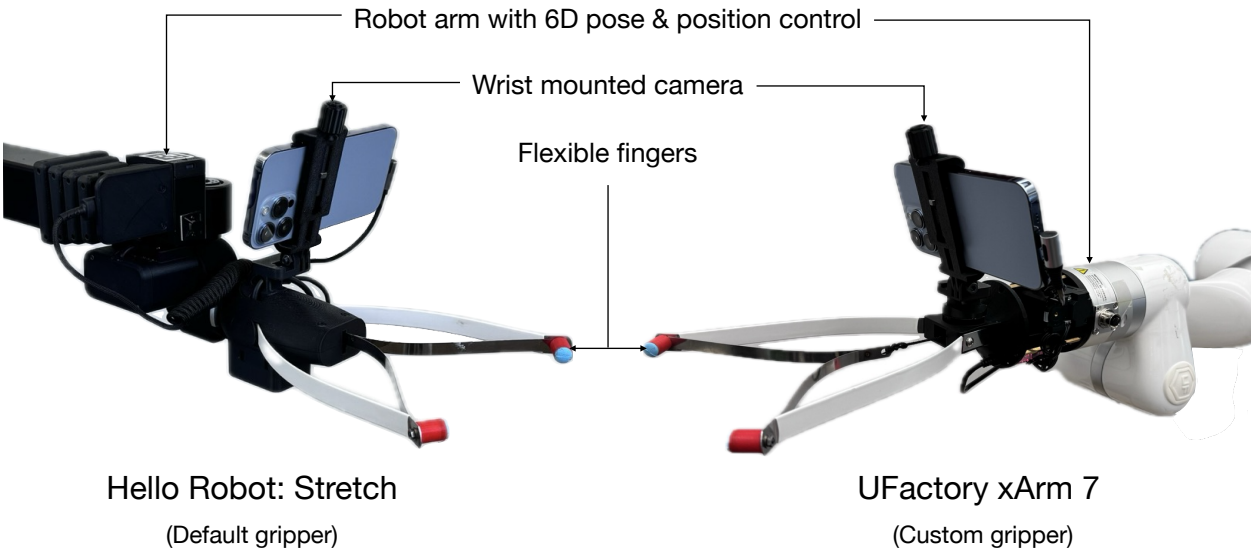


Figure 7.5: Picture of the some robot setups where our Robot Utility Models can be deployed. We show the Hello Robot: Stretch, and the xArm 7 robot with iPhone Pros on the wrist. Beyond these, we also deploy on Stretch robots with default D405 wrist cameras.

for a success detection and retrying mechanism. We define a single verification prompt for each task, and ask the mLLM to verify the success of the task given a summary of robot observations. As for the run summary, we give the mLLM every other frame from the robot camera, which is either from the head or the wrist camera depending on the task. If the mLLM detects a failure (Figure 7.4), RUM automatically resets the robot to a home position and retries the task with a new initial robot state.

7.2.5 Deployment Details

Our primary hardware for Robot Utility Models deployment is the Hello Robot: Stretch robots with an iPhone on the wrist, but we support deploying our models on any robot arm with relative 6D pose and position control (Figure 7.5). We design and release an associated robot end-effector that can be mounted on standard robot arms, such as the xArm or Franka Panda. Similarly, while we primarily use the iPhone Pro as the deployment camera, we also show deployment on other wrist cameras, such as the Intel Realsense D405, which is the default wrist camera on Hello Stretch

Edition 3 onwards. Overall, our deployment hardware system really relies on three things: our end-effector with a flexible two-fingered gripper and gripper tips, a wrist camera with a sufficient field of view, and an arm with six degrees of freedom to mount our wrist. We release default integration code for Hello Stretch 3 and an xArm wrist mount that we created, which should serve as illustrative examples for other arms.

7.3 Capabilities of Robot Utility Models

To understand the capabilities of RUMs, we evaluate each of our models on a diverse set of environments. At the same time, we try to examine our recipe for training utility models and answer a set of questions about the trained models by running a set of ablation experiments. The primary questions that we try to answer are the following:

- How well do Robot Utility Models solve a task in an unseen environment while operating on unseen objects?
- What is the relative importance of different components of Robot Utility Models, such as training data, training algorithm, and self-verification?
  - What scale of data is needed to train capable RUMs?
  - What properties of data are most important for training RUMs?
  - How does mLLM-based self-critique affect RUMs, and where does it succeed or fail?
- How well can we deploy RUMs on new robot embodiments?

Evaluation details: We set up 25 novel environments – five for each task – with objects and props not seen in the training dataset. To create these evaluation environments, we take the robot to previously unseen kitchens, purchase new furniture online (door and drawer opening), and

source new objects manually verified to not be in the training set (reorientation, bag and tissue pick up). We show sample pictures of each of the environments and objects on our Appendix E.2.3. We evaluate each system and policy for 10 trials in each of these environments, starting from the same grid of starting positions facing the task space used by [Shafiullah et al. 2023b] as we show in Appendix Figure E.1. For the retrying-based experiments, while RUMs take 1.31 tries in average to succeed (Section 7.3.6), we set a 10-try timeout to avoid getting stuck in infinite retry loops.

7.3.1 Zero-shot evaluation of RUMs on unseen environments

The most important test of capability for a Robot Utility Model is whether such a model is capable of solving the target task in a new environment operating on new objects. We test for this capability by running our RUMs on our set of 25 eval environments and objects not seen during training.

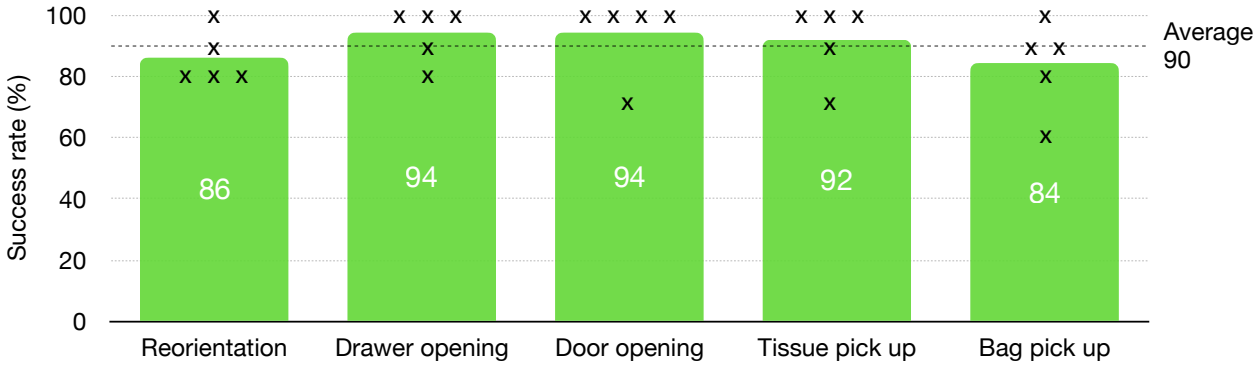


Figure 7.6: Success rate of Robot Utility Models on average over five novel scenes in five different tasks. The X's on the figure denote success rates from individual environments.

On Figure 7.6, we see that on unseen and novel environments, RUMs perform well, achieving a 90% success rate overall, and ranging between 84% to 94% on individual tasks. We discuss some of the failure cases we observe in the Appendix Section E.1.3. Additionally, we show the performance of RUMs on each test environment on Table E.1, showing that across all of our evaluation experiments, RUMs achieves some success in every environment. This success implies

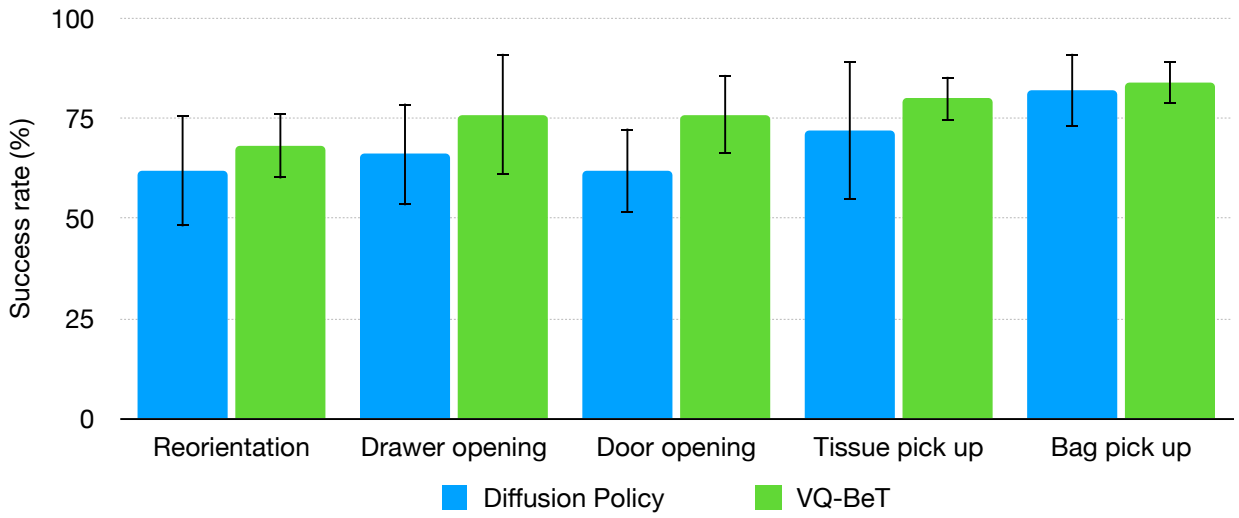


Figure 7.7: Relative comparison of the success rate (with standard error) of different policy architectures on our dataset on all five tasks without automated error correction. We see that the performance of VQ-BeT and Diffusion Policy is generally close, with VQ-BeT narrowly outperforming Diffusion Policy.

that our policies have a general idea of solving the target task; then such policies are further boosted with post-training methods (Section 7.3.6). On all of our following experiments, we try to understand these two factors separately: the raw performance of the underlying RUM policies, and the effect of introspection and retrying on the performance of RUMs.

7.3.2 Effect of policy architecture and training method on RUMs

Once we have verified that RUMs can actually solve tasks in novel environments, we investigate the relative importance of different components within the training recipe. In particular, we compare the raw performance of different policy architectures on our dataset without the introspection component. We train a set of policy classes on our datasets for each task, including VQ-BeT [Lee et al. 2024], Diffusion Policy (DP) [Chi et al. 2023], and as baselines, ACT [Zhao et al. 2023b] and MLP-BC on two of the tasks. We show the relative comparison of the base success rates of different policy architectures, without retrying, in Figure 7.7 and 7.8.

As we see in Figure 7.7, VQ-BeT and DP are the top two algorithms in terms of performance,

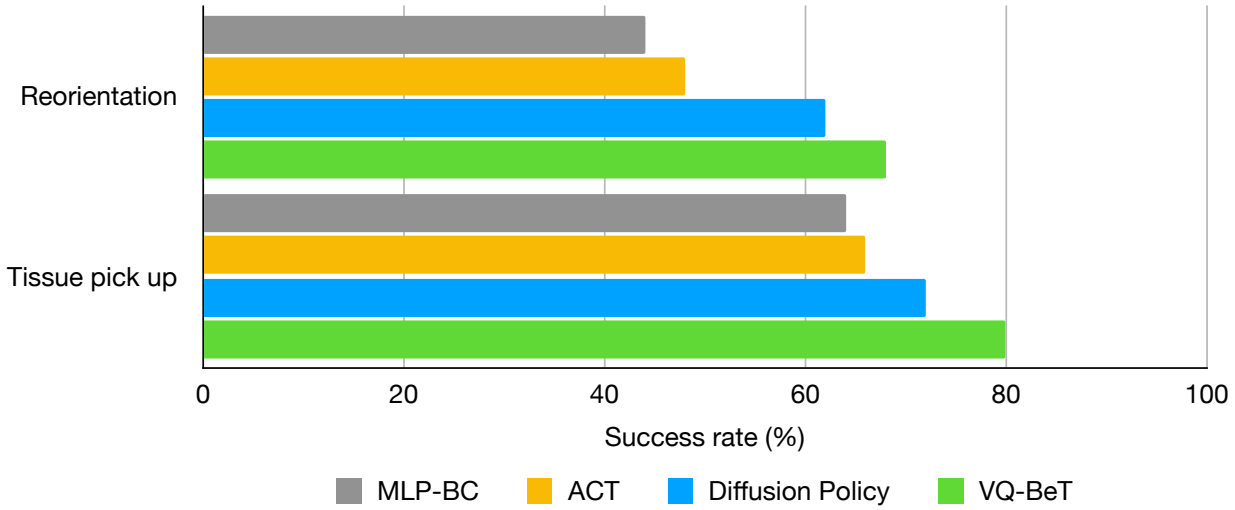


Figure 7.8: Relative comparison of different policy architectures on our dataset on two tasks without automated error correction. We see that while the performance of VQ-BeT and Diffusion Policy is generally neck-to-neck, while the performance of other algorithms is not far behind. Our experiment implies that the training data is significantly more important than training algorithm.

with comparable performance in most tasks and overlapping error bars. Moreover, we see from Figure 7.8 that while ACT and MLP-BC are not exactly on par, they are not far behind either. This observation implies that with training data of sufficient quality, the choice of algorithm may not be a make-or-break decision, and more energy should be spent on collecting diverse and accurate data. While we have similar performances on the test environment, we use VQ-BeT over DP for our final models due the higher performance and a lower latency on the robot CPU itself during deployment.

7.3.3 Effect of scaling datasets on RUMs

As our experiments show the importance of training data in creating RUMs, we investigate the properties of the dataset that a successful RUMs relies on. In particular, we dig into the scale of dataset at which reliable generalization emerges, and how RUMs' performance vary with dataset size. We train our policies on a random subset of environments from the task-specific datasets,

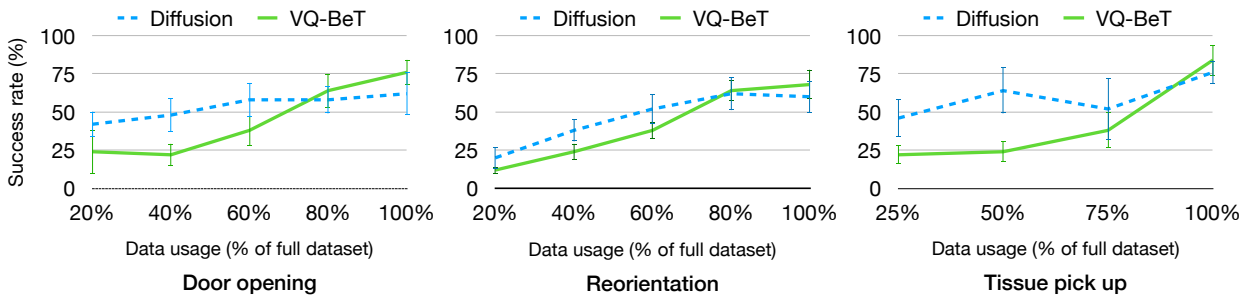


Figure 7.9: Understanding the performance change of RUMs as the dataset scales up on three of our tasks, with standard error on error bars. We see better performance from Diffusion Policy (DP) on smaller datasets, but as we scale up, VQ-BeT outperforms DP in 900–1,200 demonstrations limit.

and evaluate them on our evaluation environments.

In Figure 7.9, we show the performance of VQ-BeT and Diffusion Policy without retrying trained on such data subsets on our evaluation environments as we scale up the dataset. We see that while Diffusion Policy performs better on smaller datasets, it saturates on larger datasets where VQ-BeT outperforms it. This observation implies that while a smaller dataset may be sufficient for training a capable RUMs, a larger dataset is crucial for achieving the best performance. Even on our largest datasets, we see that the performance of VQ-BeT continues to improve as the dataset scales up, implying that more data may improve RUMs even further.

7.3.4 Importance of data diversity in training RUMs

Beyond the scale of the dataset, we also investigate how the diversity of the training data impacts the performance of RUMs in Figure 7.10 (left). We create two alternate datasets of equal size for the door opening and the object reorientation tasks. The first datasets are composed of a large number of diverse environments with roughly 25 demonstrations in each environment. The second dataset is composed of fewer, between 5 and 6, distinct environments with roughly 200 demonstrations on each environment. We see that on the door opening task, where the scene diversity is narrower, both diverse and uniform environment trained policies performed well.

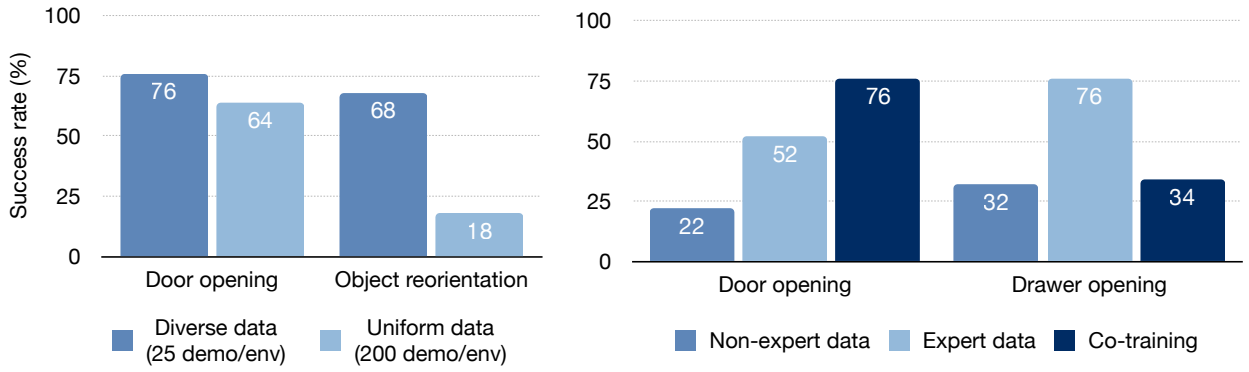


Figure 7.10: Understanding the importance of different qualities of data in training RUMs. On the left, we see that diverse datasets are more valuable than more uniform datasets, with strong effects on the reorientation task with many unseen environments and object. On the right, we see that usually expert data is more valuable than non-expert or play data while learning behavior on a same sized dataset. Moreover, we see that co-training with expert data and play data may sometimes reduce the policy performance, contrary to common knowledge.

However, in the reorientation task, with many different unseen environments and objects, only diverse-environment trained RUM policy performs well – the policy trained on more uniform environments experiences a 50% performance drop. This result implies that to train an effective RUM, collecting a diverse dataset is important.

7.3.5 Impact of using expert demonstrations on training policies

While scaling up the dataset size and diversity is important for training RUMs, an important question to consider is the quality of the training dataset. Namely, while it may be easy to collect a large number of demonstrations by a large number of demonstrators, the quality of the demonstrations may vary. In this section, we investigate the value of using expert demonstrations in training RUMs.

In Figure 7.10 (right) we compare the performance of RUMs trained on roughly 500 demonstrations, where the data is either sampled from expert or non-expert demonstration collectors. Here, "expertise" is defined as experience deploying Dobb·E policies on the robot. We see that in general, expert data is more valuable than non-expert data, with expert data outperforming non-expert

data in all tasks. Moreover, we see that co-training with expert and non-expert data can sometimes, but not always, improve the performance of the policy. This observation implies depending on the task, data quality can have different levels of suboptimality, and in extreme cases may even hurt performance in co-training, which goes against a common practice in some earlier works [Zhao et al. 2023b; Khazatsky et al. 2024].

7.3.6 Effects of introspection and retrying with self-critique in RUMs

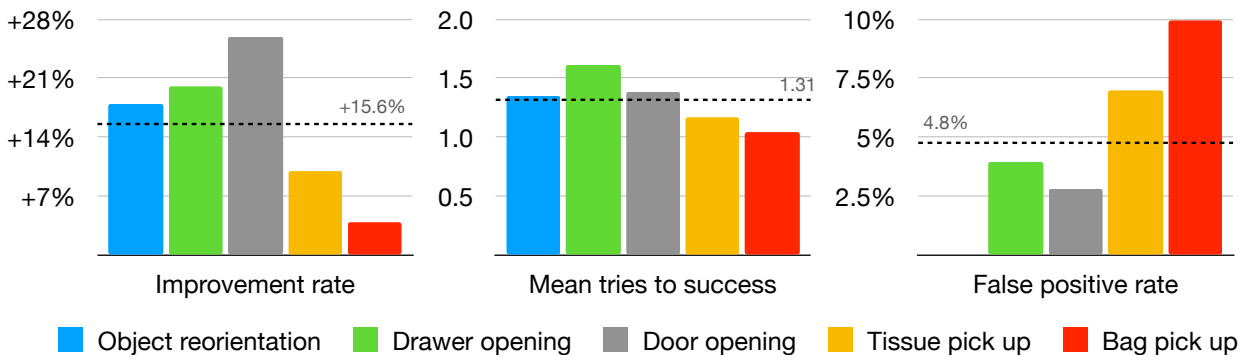


Figure 7.11: Understanding the details of introspection and retrying in RUMs. On the left, we see that retrying improves the performance of RUMs significantly, with an average 15.6% improvement. In the middle, we see that with retrying, most tasks get solved quite fast, on average with 1.31 tries. On the right, we see that while the mLLM is able to help, it can also have false positives (4.8% average over five tasks) which may let some errors slip past.

In RUMs, we are using a multimodal large language model (mLLM) as a self-critique method to identify failures. However, a pretrained mLLM in practice is just another layer of fail-safe for our robot deployment, and not a guarantee of success in itself. Thus, in this section we try to understand how it helps, and how such introspection method can fail.

In Figure 7.11 (left), we can see the improvement rate of using self-critique over simply using the RUM policies without any retrying mechanism. On average over our 5 tasks, we see a 15.6% improvement over simply using RUM policies. While retrying is crucial to a higher success rate, a system that is stuck retrying for a long time is much less useful. Thankfully, on average,

when RUMs succeeds, it does so within 1.31 tries on average, as we see from Figure 7.11 (middle). Finally, we analyze the primary failure mode of mLLMs, which is predicting false positives: classifying a trajectory as a success when it's actually a failure. On average, 4.8% of our trajectories exhibit such behavior, constituting of half of the total errors, as seen on Figure 7.11 (right).

7.3.7 Transferring RUMs to different embodiments

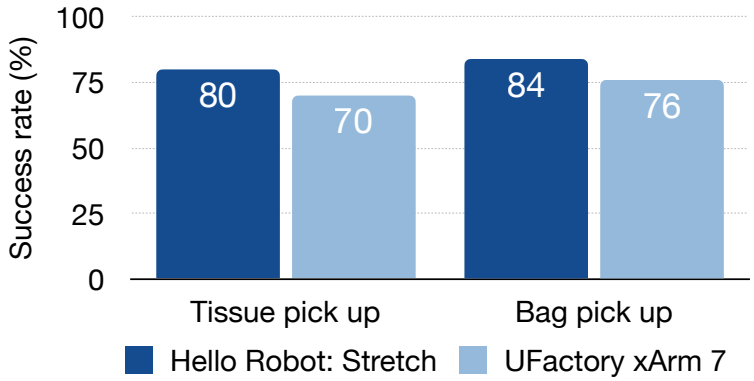


Figure 7.12: Performance of RUMs without corrections on different embodiments as shown in Figure 7.5: RUMs can transfer to different embodiments with minimal loss in performance.

Finally, we investigate the ability of RUMs to be transferred to different embodiments and cameras. We test the performance of two RUMs on the other robot setup shown in Figure 7.5: UFactory xArm 7, which is different from the Hello Robot Stretch setup we run other experiments on. We see that RUMs can be transferred to different embodiments and cameras with minimal loss in performance: roughly 10% drop in performance in both cases without corrective mLLM feedback, as shown in Figure 7.12. We expect combining RUMs with the mLLM self-critique would result in similar increase in performance in other embodiments as well; in fact, with an external third person camera, we expect to see a higher portion of the errors being caught and corrected. This experiment implies that RUMs can be easily deployed on different robots and cameras with minimal effort, making it a versatile tool for a wide range of robotic applications.

7.4 Related works

Large Scale Data Collection: The data acquisition pipeline represents one of the most critical element of a data-driven robot learning framework. Previous works has employed a diverse array of data acquisition techniques, combining many open-sourced datasets across diverse simulation or real-world data including diverse robot embodiment from many institutions across the globe [Reed et al. 2022; Padalkar et al. 2023; Zitkovich et al. 2023; Khazatsky et al. 2024].

The most common approaches to robot demonstration collection involves pairing the robot or end-effector with remote controller devices or kinematically isomorphic equipment. The devices utilized have a range of complexity and forms: they encompass full robotic exoskeletons [Zhao et al. 2023a; Ishiguro et al. 2020; Fang et al. 2023a], as well as simpler data collection tools [Zhao et al. 2023b; Wu et al. 2023; Fu et al. 2024b], and also methods that don't require physically moving a robot [Shafiullah et al. 2023b; Song et al. 2020; Pari et al. 2021; Young et al. 2020; Chi et al. 2024]. Additionally, various control methods have also been employed, including the use of video game controllers [Liu et al. 2024a; Sian et al. 2004], Virtual Reality (VR) devices [Iyer et al. 2024; Cui et al. 2022; Cheng et al. 2024; Yang et al. 2024b; Park and Agrawal 2024; Arunachalam et al. 2023b,a; Fu et al. 2024a], and mobile phones [Mandlekar et al. 2018].

While the most intuitive method is to physically move a real robot, it is both difficult to do and hard to scale to a diverse set of environments. The hardware controller approach can be inefficient because it requires the demonstrator to mentally map robot behavior to controller inputs. The opposite, using a device without moving the robot is efficient in that the demonstrator's movements can be mapped directly to the robot, but it is challenging to apply force feedback. Studies that provides perspective on the relative merits of these two direction are [Shafiullah et al. 2023b; Chi et al. 2024], which combines the versatility of simple controller with the intuitiveness of moving a physical end-effector. In this work, we employ a device that inherits and improves the device

proposed from [Shafiullah et al. 2023b; Chi et al. 2024] for our data collection pipeline.

Pretrained Robot Models: Pre-trained foundation models have demonstrated a wide range of generalization performance across various domains, with the capability to learn from internet-scale pre-training data [Devlin et al. 2018; Radford et al. 2021; Dubey et al. 2024; Kirillov et al. 2023]. However, in comparison to these vision and language pre-trained models, learning a foundation model for robotics has been considered a relatively challenging area, due to the limited quantity of available datasets [Kappler et al. 2015; Levine et al. 2016; Depierre et al. 2018; Zhu et al. 2023], the significant discrepancy across the domains [Dasari et al. 2019; Kalashnikov et al. 2021; Padalkar et al. 2023], and the inherently challenging nature of the action datasets in terms of tokenization [Lee et al. 2024; Zitkovich et al. 2023; Zheng et al. 2024].

To address these issues, recent research is increasingly adopting techniques that introduce modular and hierarchical systems, incorporate pre-trained language and visual models [Li et al. 2023; Nair et al. 2022b; Karamcheti et al. 2023; Shafiullah et al. 2023a; Liu et al. 2024c; Gupta et al. 2024], and collect large scale data with efficient data collection schemes [Khazatsky et al. 2024; Zitkovich et al. 2023; Walke et al. 2023; Ebert et al. 2022; Fang et al. 2023b]. Consequently, they have enabled the pre-trained foundation robot models to exhibit enhanced generalization performance, thereby showcasing that the robotic agents are capable of operating in more than one robot embodiment and operating environment [Team et al. 2024; Reed et al. 2022; Kim et al. 2024; Doshi et al. 2024]. In contrast with the aforementioned approaches, which follow a method of training on internet-scale data and fine-tuning on task-specific data, our approach does not expect that the model will have access to a dataset in the environments where the robot is expected to operate. Rather, this project demonstrates the capacity of generalizable performance without a necessity to fine-tune the model for each novel robot embodiment and environment.

Large Models Feedback and Improvement: Due to their capacity to comprehend intricate semantics and relations, Natural language and Large language models (LLM), have recently been applied to robotic agents powered by imitation learning [Fried et al. 2018; Kim et al. 2024; Shridhar et al. 2022; Jang et al. 2021] and reinforcement learning [Du et al. 2023; Goyal et al. 2021].

Among the wide capabilities afforded by language models, those commonly employed in the context of decision-making include providing feedback in the resolution of uncertain information [Ren et al. 2023; Mullen Jr and Manocha 2024; Huang et al. 2022b; Liu et al. 2023c; Guo et al. 2023; Park et al. 2023; Gao et al. 2024], suggesting affordance of what is possible in the environments by combining with Value functions [Brohan et al. 2023b], and imagination of outcomes [Zhang et al. 2024] or planning and decompose complex tasks into mid-level plans [Song et al. 2023; Huang et al. 2022a; Zeng et al. 2022; Sharma et al. 2021]. Language models could also be used to improve the overall performance of autonomous agent systems by improving reward signal [Nair et al. 2022a; Goyal et al. 2021; Ma et al. 2023], leveraging their long-horizon reasoning [Dalal et al. 2024; Zhou et al. 2023a; Blukis et al. 2022], or designing environments [Ma et al. 2024]. In this project, we employ the mLLM to provide feedback in the form of a reset signal in open-ended environments, a manner analogous to that of the studies above.

7.5 Limitations

While in this work we create Robot Utility Models that can perform particular tasks zero-shot in novel environments, there are certain limitations that future versions can improve upon. The primary limitation that we see are of hardware: for example, two-fingered grippers like our Stick-v2 are unable to open doors with round doorknobs. Similarly, while flexible fingertips can be more lenient for the policy, it makes it hard to manipulate heavy objects. We encourage more research on better gripper and fingertip design to address these issues. Secondly, we assume navigation to be a separate component, and in this work assume that the robot is in the task space

facing the task objective. Combining with modular navigation work such as [Liu et al. 2024c] should address this issue. Finally, for mLLM introspection and retrying, we assume that the errors made by our model (a) leaves the task-space somewhat in-distribution, and (b) allows for an easy reset of the robot to the initial state. Increasing training data with failure recovery behavior in our dataset should let our robots recover more naturally from such failure cases.

Postscript

Robot Utility Model is the first work that went beyond my nearest neighbor theory of behavior cloning – if the robot is performing a complex task in an entirely new environment, it must have to do something beyond nearest neighbor. This method may be discovering a representation that looks beyond the environment dependent features.

One of the underappreciated features of this work is the potential economic value of being able to do one narrowly defined task well in a variety of environment and objects. Many large behavior models currently define their task interface poorly – it is hard to specify what tasks and in what environments the system would be able to solve. It may be more prudent to instead build up from a select set of tasks that we can be fairly confident that the model can do under most circumstances, and think about different permutations of behaviors from the ground up. Finally, RUMs gives us a recipe to build a "minimal" generalist robot model. This recipe can now be used to study and understand generalization behaviors for robot models, and hopefully used to discover new principles that can make such policies more efficient and effective in the real world.

Acknowledgements

This work was co-led with Haritheja Etukuru, and co-authored with Norihito Naka, Zijin Hu, Seungjae Lee, Julian Mehu, Aaron Edsinger, Chris Paxton, Soumith Chintala, and Lerrel Pinto. We thank Shenglong Wang and the NYU HPC team for helping us with compute, Blaine Matulevic and Binit Shah for supporting our hardware needs, and Siddhant Haldar and Jeff Cui for providing feedback on the paper. NYU authors are supported by grants from Honda, Hyundai, NSF award 2339096 and ONR awards N00014-21-1-2758 and N00014-22-1-2773. MS is supported by the Apple Fellowship. LP is supported by the Packard Fellowship. SL is supported by the Daishin Songchon Foundation. Hello Robot authors are supported by NIH NIA R43AG072982.

8 | Building an Open-source Bimanual Mobile Robot for Generalizable Robotics: Cone-E

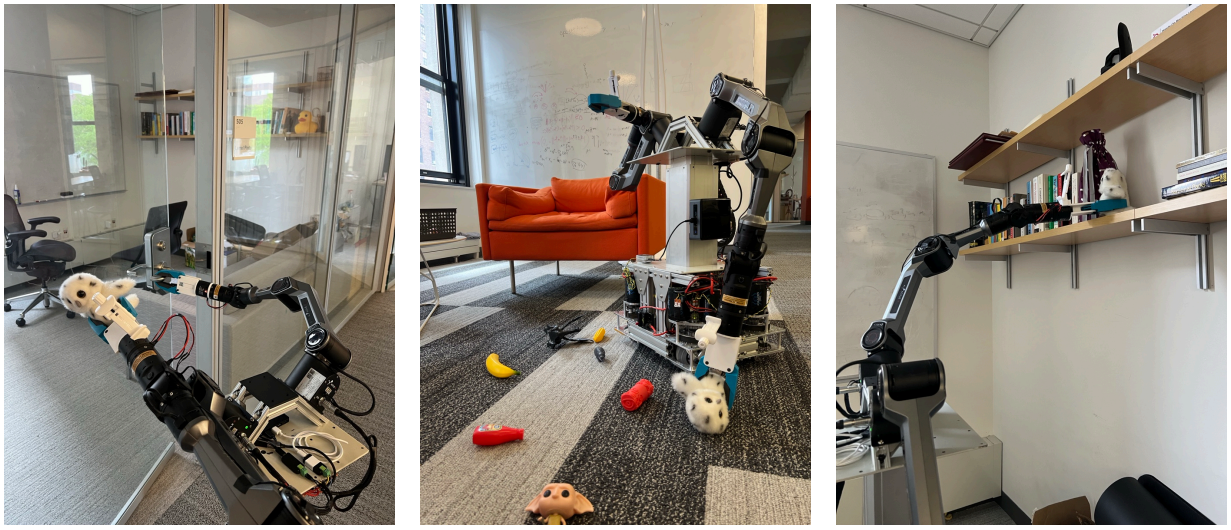


Figure 8.1: Cone-E is an open-source, bimanual mobile manipulator designed as a general-purpose research platform.

8.1 Introduction

Applications of machine learning in robotics have made tremendous progress in recent years in robot navigation [Sridhar et al. 2024; Liu et al. 2024b; Yang et al. 2024a; Gervet et al. 2023a; Shafiullah et al. 2023a], locomotion [Rudin et al. 2022; Agarwal et al. 2023; Cheng et al. 2023; Fu et al. 2022; Margolis et al. 2022], and manipulation [Zitkovich et al. 2023; Chi et al. 2023; Kim et al. 2024; Haldar et al. 2024; Zhao et al. 2023b; Lin et al. 2024]. Such advances in robotics have been supported by accessible, low cost hardware such as the Unitree A1 and G1 robots, the Hello Robot: Stretch, or the Mobile Aloha open-source bimanual manipulator [Fu et al. 2024b]. However, there is a noticeable gap in the currently available accessible platforms for mobile manipulation, in particular for bimanual mobile robots. Currently, such available platforms on the market tend to be inaccessible, hard to build upon, or have limited functionality due to, respectively, high-cost, closed source design, and hardware limitations.

In this work, we propose a new mobile manipulator design to accelerate generalizable robotics research – aiming to provide a reliable platform that will fuel indoor mobile manipulation research. Our most important considerations for this platform are to make it low-cost, and easy to build and repair with off-the-shelf parts, and easy to control in various ways. We identify some key needs for a good mobile manipulation research platform: dexterous bimanual arms, an omnidirectional base, and a vertically extended workspace that reaches both the floor and overhead. Beyond hardware capabilities, we identify quality-of-life developments for researchers, like having a long battery life, a small footprint, and a readable, fully open source software stack.

Our proposed mobile manipulator, Cone-E, is low-cost (with a bill of material cost $12-13K USD) and fully integrated to provide whole-body manipulation. With the publication of this work, we will open source the hardware design, including a BOM and an assembly guide, the full controller software stack, and a suite of our general-purpose “utility” policies. We believe our design will

propel further research into whole-body and mobile manipulation by providing access to a stable platform with minimal dynamic constraints.

8.2 Hardware Design

In this section, we discuss how Cone-E achieves the hardware design goals identified in Section 8.1.

8.2.1 Mobile Base

Cone-E has an omnidirectional base to allow flexible navigation, intuitive teleoperation, and simplified policy learning. Many current commercial robots, such as the Hello Robot: Stretch [Kemp et al. 2022] or the Rainbow RB-Y1 [Rainbow Robotics 2025], use a differential-drive base due to its simplicity and lower cost. While cheap, this type of drive is non-holonomic, meaning the state of the system is dependent on the path taken in order to achieve it.

Differential-drive constraint limits arbitrary position control which is important for closed-loop learned policies. Therefore, we designed our base as a swerve drivetrain with four wheels. We use readily available components from the FIRST Robotics Competition (FRC) ecosystem [FIRST (For Inspiration and Recognition of Science and Technology) 2024], similar to Tidybot++ [Wu et al. 2024]. A frame made of aluminum extrusions carry the four swerve modules, a power distribution block and the battery.

Unlike TidyBot++, we do not modify the swerve modules to create caster wheels. Our base is *non-holonomic* if modeled at the level of infinitesimally small timesteps. However, an abstraction of the system with discrete timesteps, longer than the steering duration, still renders a holonomic system. We find the maneuverability of swerve modules is enough for non-dynamic tasks in household environment while bypassing additional build complexity from machining caster modules.

We further refine our design to give the base a small footprint (34 x 42 cm) allowing navigation

in household environments similar to humans. We designed this base to be more compact than TidyBot++ by simplifying the electrical circuitry and running all digital components from a single 24V 20Ah NMC battery. NMC batteries have higher power density compared to SLA and LiFePO4 batteries, providing long runtime in a compact form factor. Our base motors, the lift motor and the arms all run on 24V and can be directly powered from the power distribution block. The control module, an Intel NUC mini PC, runs on 19V and needs a step-down voltage regulator after the power distribution block. Practical splice connectors like Wago [WAGO Kontakttechnik GmbH & Co. KG 2025] and power distribution panel with many output channels keep the circuitry easy to build and customizable. In addition to the circuitry, we use SDS MK4c swerve modules instead of SDS MK4 due to their smaller footprint and lower chassis mounting.

8.2.2 Telescoping Lift

A core component to increase the vertical reach of mobile manipulators is a lift mechanism that provides a vertical degree of freedom. Lift mechanisms allow the robot to raise or lower its torso consisting of arm and sensors and expands its workspace beyond a fixed mounting height. This added vertical mobility is essential for household tasks like reaching high shelves, picking objects from the floor, or achieving optimal sensor viewpoints.

Most commercial lift designs are custom built per order and expensive, not readily available on the market. To keep our design low-cost and easy to build with the off-the-shelf parts, we re-purpose an adjustable height telescoping table (shown in Figure 8.2) as our robot lift. The table is a telescoping lead screw mechanism driven manually with a hand crank, which we motorize for our purposes. Inside, there are three lead screws that nest inside each other. The screws rotate in sequence. This allows for compact collapsed length and extended height adjustment. The screws are self-locking, thus, they hold position when unpowered. This makes Cone-E more efficient as the lift does not need to draw power when stationary. The outside of the lift are three telescoping

aluminum columns that are held together by rubber friction pads. We use the thin steel panels on the top and bottom of the lift to mount the torso and attach the lift onto the base respectively.

We automate this table by motorizing the hand crank drive shaft. We create a timing belt pulley that fits on the shaft using nylon or metal 3-D printing. Then, using a timing belt and a BLDC servo motor, we can control the lift height. The lift is 30.5 cm at its lowest and 72 cm at its highest. We find the 41.5 cm stroke length to be enough for being able to reach the ground and also doing tabletop manipulation on high surfaces. In addition to providing extended reach, the lift acts as an extra degree of freedom that we can utilize in our inverse kinematics solver.

We use the integrated encoders inside the motor as a feedback to compute lift position. The lead screws inside the lift have a $6mm$ thread pitch. We use a 60-teeth and 18-teeth pulley on the lift and motor shaft respectively. To move the lift through its full-range, the motor needs to rotate approximately 225 rotations. To calibrate, the lift needs to "home" when the robot is turned on.

8.2.3 Arms and the Gripper

We build Cone-E as a bimanual robot that supports two 6DOF arms and custom grippers as manipulation tools. We choose AgileX Piper arms due to their low-cost (only $2,500) and light weight (4.2 kg). The arms are mounted onto an extruded aluminium torso with 45-degree shoulders. We choose this shape to balance the arms' forward and downward reach. The angled shoulders also prevent the elbows of the arms from colliding with each other even if their mounting points are close.

As the end-effector on the arm, we choose the NYU gripper introduced in [Etukuru et al. 2024]. The angular jaw design allows both precise manipulation and large force application. As an end-effector camera, we use an iPhone following the same work, and for data collection use the hand-held version of the NYU gripper with the mounted iPhone and the associated app, AnySense. The app records video, high-quality $SE(3)$ pose, and any supplementary information streamed

over bluetooth, all at 30fps. We designed this tool to be more ergonomic and compact compared to other similar tool designs such as [Etukuru et al. 2024; Shafiullah et al. 2023b; Chi et al. 2024].

8.2.3.1 Compliant Controller

A compliant controller is essential to absorb unexpected forces encountered during manipulation and ensure safety in learned policy deployments. Therefore, we implement a joint stiffness controller with two layers. The low-level real time controller runs at 200 Hz, while the policy sets targets for this controller at much lower frequencies. The typical joint stiffness controller objective is

$$-\tau_g(q) + K_p(q_{ref} - q) + K_d(\dot{q}_{ref} - \dot{q})$$

where q is the measured joint positions and q_{ref} is the target position set by the upstream controller. The system acts like a spring-damper around the reference position with stiffness coefficient K_p and damping coefficient K_d. The feedforward torque gravity compensation allows us to set stiffness gains lower, resulting in compliant movement.

Figure 8.2: Cone-E is modular and easily customizable with different arms, end-effectors and sensors.

8.3 Applications of Cone-E

8.3.1 Teleoperation

We teleoperate Cone-E using a Quest VR device following [Iyer et al. 2024]. The VR controllers are re-mapped to robot control in the following way. We re-target the left and right controller poses to the corresponding arm's end-effector pose. The left and right joysticks on the controllers are used to command rotational and translational velocities to the base in the planar $SE(2)$ workspace respectively. The trigger buttons on the joysticks are used to control the lift height. Quest controller commands are published to the robot mini PC over WiFi. We find 30 Hz to be the ideal VR command frequency to balance robot responsiveness against network delays.

8.3.2 Policy Learning

Following Etukuru et al. [2024], we used our hand-held data collection tool with an iPhone Pro to collect demonstrations for a general pick-up task. Our portable hand-held tool enables us to collect demonstrations in diverse environments. We collected approximately 5,000 demonstrations to train a general pick-up policy. We use a VQ-BeT [Lee et al. 2024] model with 30M parameters, which runs entirely on the CPU of Cone-E's mini PC. The pick-up model predicts the $SE(3)$ relative action in the current end-effector frame and the absolute gripper pose. This end-effector pose is then fed to our arm differential inverse kinematics controller, which calculates the next joint positions for the robot.

The policy takes in camera observations and predicts new actions at 2Hz, predicting the desired end-effector pose. In contrast, our low-level joint stiffness controller runs at 200Hz. To bridge this frequency gap and ensure smooth motion, we interpolate the joint commands to reach the target pose within 1 second. The policy issues a new command when the preceding one is halfway

completed (every 0.5s), thereby enabling continuous and smooth robot control.

8.4 Limitations

In this work, we introduce Cone-E, an open-source bimanual mobile manipulator robot platform. While we believe it offers a great balance between cost and functionality, there are certain affordances, such as a head camera and twisting neck and torso, that are not present in the current version. By open sourcing our design, we hope that the community can customize the platform to their needs while iterating on future such platforms in an open and collaborative way.

Postscript

One of the primary limitation in the current era of robot learning is not about learning – it is that we do not have good, unified platforms to run our experiments. Being involved in the Cone-E project highlighted all the consideration that goes into designing a robot specifically for learning purposes. Similarly, Cone-E is a study in understanding the trade-off in robot building while on a budget, and hopefully will inspire future works in the same vein building open-source hardware that makes open robot learning research more expedient.

Acknowledgements

This work was led by Enes Erciyes, co-authored with Haritheja Etukuru and Soumith Chintala, and advised by Lerrel Pinto.

Part III

Semantic Memory for Long-horizon Intelligence

9 | Weakly Supervised Semantic Fields for Robotic Memory: CLIP-Fields

9.1 Introduction

In order to perform a variety of complex tasks in human environments, robots often rely on a spatial semantic memory [Blukis et al. 2022; Min et al. 2021; Gervet et al. 2023a]. Ideally, this spatial memory should not be restricted to particular labels or semantic concepts, would not rely on human annotation for each scene, and would be easily learnable from commodity sensors like RGB-D cameras and IMUs. However, existing representations are coarse, often relying on a preset list of classes and capturing minimal semantics [Blukis et al. 2022; Gervet et al. 2023a]. As a solution, we propose CLIP-Fields, which builds an implicit spatial semantic memory using web-scale pretrained models as weak supervision. Recently, representations of 3D scenes via neural implicit mappings have become practical [Sucar et al. 2021; Sitzmann et al. 2019]. Neural Radiance Fields (NeRFs) [Mildenhall et al. 2020], and implicit neural representations more generally [Ortiz et al. 2022] can serve as differentiable databases of spatio-temporal information that can be used by robots for scene understanding, SLAM, and planning [Li et al. 2022; Simeonov et al. 2022; Chen et al. 2022b; Driess et al. 2022; Ortiz et al. 2022].

Concurrently, web-scale weakly-supervised vision-language models like CLIP [Radford et al.

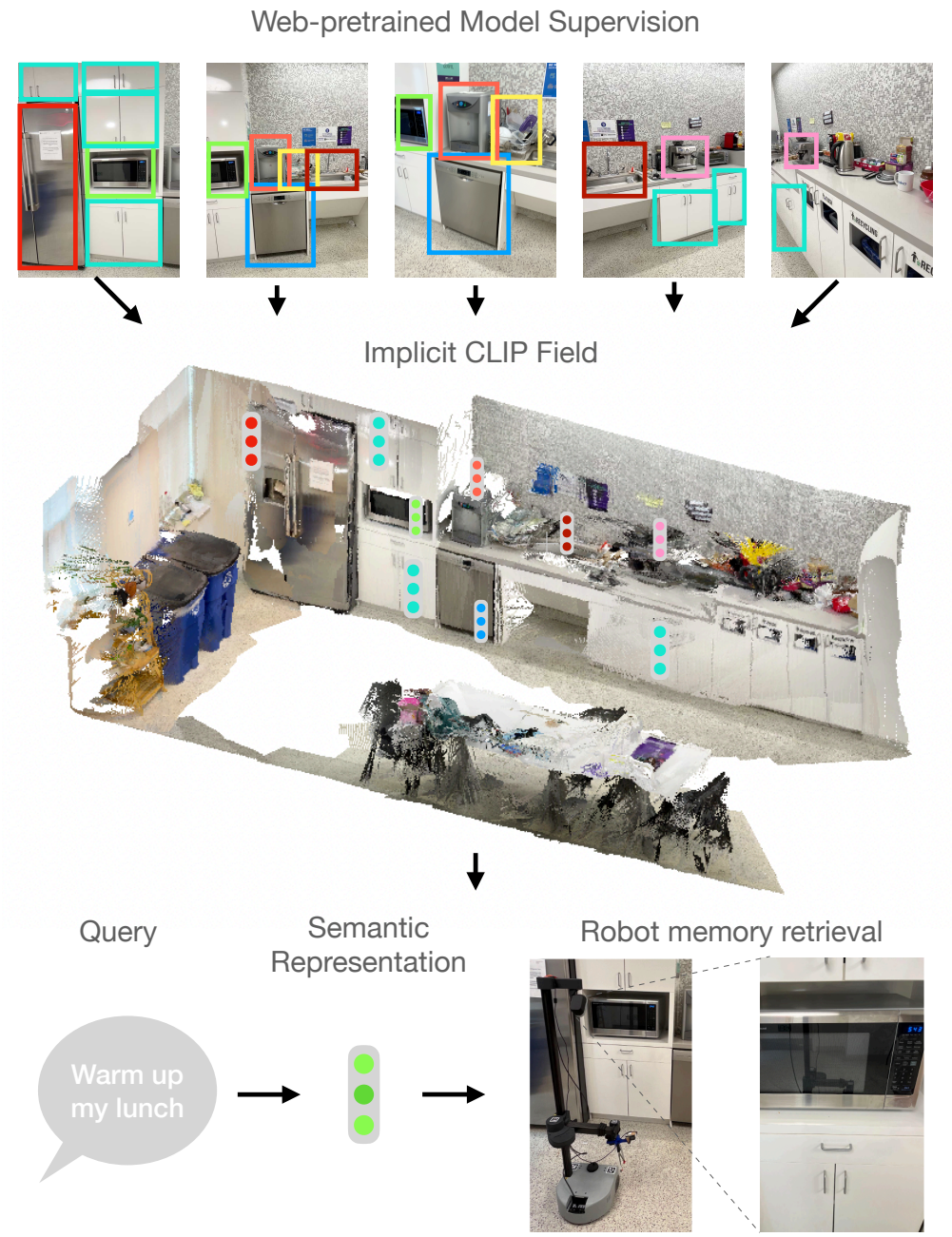


Figure 9.1: Our approach, CLIP-Fields, integrates multiple views of a scene and can capture 3D semantics from relatively few examples. This results in a scalable 3D semantic representation that can be used to infer information about the world from relatively few examples and functions as a 3D spatial memory for a mobile robot.

2021] have shown that the ability to capture powerful semantic abstractions from individual 2D images. These have proven useful for a range of robotics applications, including object

understanding [Thomason et al. 2022] and multi-task learning from demonstration [Shridhar et al. 2022]. Their applications have been limited, however, by the fact that these trained representations assume a single 2D image as input; it is an open question how to use these together with 3D reasoning.

In this work, we introduce a method for building weakly supervised semantic neural fields, called CLIP-Fields, which combines the advantages of both of these lines of work. CLIP-Fields is intended to serve as a queryable 3D scene representation, capable of acting as a spatial-semantic memory for a mobile robot. We show that CLIP-Fields is capable of open-vocabulary segmentation and object navigation in a 3D scene using only pretrained models as supervision.

Our key idea is to build a mapping from locations in space $g(x, y, z) : \mathbb{R}^3 \rightarrow \mathbb{R}^d$ that serves as a generic differentiable spatial database. This dataset is trained to predict features from a set of off-the-shelf vision-language models trained on web-scale data, which give us weak supervision. This map is trained on RGB-D data using a contrastive loss which encourages similarity between features predicted at specific spatial locations.

Thus, from the point of view of a robot using CLIP-Fields as a spatial database for scene-understanding, training g itself can be entirely self-supervised: the full pipeline, including training the underlying image models, need not use any explicit supervision. On the other hand, as we show in our experiments, even without any explicit supervision, the spatial database g can naturally capture scene-specific information.

We demonstrate our method on tasks such as instance segmentation and identification. Furthermore, we give qualitative examples of image-view localization, where we need to find the spatial coordinates corresponding to an image and localizing text descriptions in space. Finally, we demonstrate CLIP-Fields on a real robot by having the robot move to look at various objects in 3D given natural language commands. These experiments show how CLIP-Fields could be used to power a range of real-world applications by capturing rich 3D semantic information in an

accessible way.

9.2 Related work

Vision-Language Navigation. Much recent progress on vision-language navigation problems such as ALFRED [Shridhar et al. 2020] or RXR [Ku et al. 2020] has used spatial representations or structured memory as a key component to solving the problem [Min et al. 2021; Blukis et al. 2022; Wang et al. 2021; Gadre et al. 2022]. HLSM [Blukis et al. 2022] and FiLM [Min et al. 2021] are built as the agent moves through the environment, and rely on a fixed set of classes and a discretization of the world that is inherently limiting. By contrast, CLIP-Fields creates an embedding-dependant implicit representation of a scene, removing dependency on a fixed set of labels and hyperparameters related to environment discretization. Other representations [Wang et al. 2021] do not allow for 3D spatial queries, or rely on dense annotations, or accurate object detection and segmentation [Gadre et al. 2022; Chen et al. 2020b; Azuma et al. 2022].

Concurrently with our work, NLMap-SayCan [Chen et al. 2022a] and VLMaps [Huang et al. 2023b] proposed two approaches for real-world vision-language navigation. NLMap-SayCan uses a 2D grid-based map and a discrete set of objects predicted by a region-proposal network [Chen et al. 2022a], while CLIP-Fields can make predictions at different granularities. VLMaps [Huang et al. 2023b] use a 2D grid-based representation and operate on a specific, pre-selected set of object classes. By contrast, CLIP-Fields can operate on 3D data, allowing the agent to look up or down to find objects. All three methods assume the environment has been explored, but both [Chen et al. 2022a] and [Huang et al. 2023b] look at predicting action sequences, while we focus on the problem of building an open-vocabulary, queryable 3D scene representation.

Pretrained Representations. Effective use of pretrained representations like CLIP [Radford et al. 2021] seems crucial to deploying robots with semantic knowledge in the real world. Recent works

have shown that it is possible to use supervised web image data for self-supervised learning of spatial representations. Our work is closely related to [Chaplot et al. 2021], where the authors show that a web-trained detection model, along with spatial consistency heuristics, can be used to annotate a 3D voxel map. That voxel map can then be used to propagate labels from one image to another. Other works, for example [Datta et al. 2022], use models specifically trained on indoor semantic segmentation to build semantic scene data-structures.

Cohen et al. [Cohen et al. 2022] looks at personalizing CLIP for specific users and rare queries, but does not build 3D spatial representations conducive to robotics applications, and instead functions on the level of individual images.

Implicit Representations. There is a recent trend towards using NeRF-inspired representations as the spatial knowledge base for robotic manipulation problems [Simeonov et al. 2022; Driess et al. 2022], but so far this has not been applied to open-vocabulary object search. As in [Zhi et al. 2021; Sucar et al. 2021; Vora et al. 2021; Kobayashi et al. 2022; Tschernezki et al. 2022], we use a mapping (parameterized by a neural network) that associates to an (x, y, z) point in space a vector with semantic information. In those works, the labels are given as explicit (but perhaps sparse) human annotation, whereas, in this work, the annotation for the semantic vector are derived from weakly-supervised web image data.

Language-based Robotics. Several works [Shridhar et al. 2022; Thomason et al. 2022] have shown how features from weakly-supervised web-image trained models like CLIP [Radford et al. 2021] can be used for robotic scene understanding. Most closely related to this work is [Ha and Song 2022], which uses CLIP embeddings to label points in a single-view 3D space via back-projection. In that work, text descriptions are associated with locations in space in a two step process. In the first step, using an ViT-CLIP attention-based relevancy extractor, a given text description is localized in a region on an image; and that region is back-projected to locations in space (via depth information). In the second step, a separately trained model decoupled from

the semantics converts the back-projected points into an occupancy map. In contrast, in our work, CLIP embeddings are used to directly train an implicit map that outputs a semantic vector corresponding to each point in space. One notable consequence is that our approach integrates semantic information from multiple views into the spatial memory; for example in Figure 9.6 we see that more views of the scene lead to better zero-shot detections.

9.3 Background

In this section, we provide descriptions of the recent advances in machine learning that makes CLIP-Fields possible.

Contrastive Image-Language Pretraining This pretraining method, colloquially known as CLIP [Radford et al. 2021], is based on training a pair of image and language embedding networks such that an image and text strings describing that image have similar embeddings. The CLIP model in [Radford et al. 2021] is trained with a large corpus of paired image and text captions with a contrastive loss objective predicting which caption goes with which image. The resultant pair of models are able to embed images and texts into the same latent space with a meaningful cosine similarity metric between the embeddings. We use CLIP models and embeddings heavily in this work because they can work as a shared representation between an object's visual features and its possible language labels.

Open-label Object Detection and Image Segmentation Traditionally, the objective of object detection and semantic segmentation tasks has been to assign a label to each detected object or pixels. Generally, these labels are chosen out of a set of predefined labels fixed during training or fine-tuning. Recently, the advent of open-label models have taken this task to a step further by allowing the user to define the set of labels during run-time with no extra training or fine-tuning.

Such models instead generally predict a CLIP embedding for each detected object or pixel, which is then compared against the label-embeddings to assign labels. In our work, we use Detic [Zhou et al. 2022] pretrained on ImageNet-20k as our open-label object detector. We take advantage of the fact that besides the proposed labels, Detic also reports the CLIP image embedding for each proposed region in the image.

Sentence Embedding Networks for Text Similarity CLIP models are pretrained with image-text pairs, but not with image-image or text-text pairs. As a result, sometimes CLIP embeddings can be ambiguous when comparing similarities between two images or pieces of texts. To improve CLIP-Fields' performance on language queries, we also utilize language model pretrained for semantic-similarity tasks such as Sentence-BERT [Reimers and Gurevych 2019]. Such models are pretrained on a large number of question-answer datasets. Thus, they are also good candidates for generating embeddings that are relevant to answering imperative queries.

Neural Fields Generally, Neural Fields refer to a class of methods using coordinate based neural networks which parametrize physical properties of scenes or objects across space and time [Xie et al. 2022]. Namely, they build a map from space (and potentially time) coordinates to some physical properties, such as RGB color and density in the case of neural radiance fields [Mildenhall et al. 2020], or a signed distance in the case of instant signed distance fields [Ortiz et al. 2022]. While there are many popular architectures for learning a neural field, in this paper we used Instant-NGP [Müller et al. 2022] as in preliminary experiments we found it to be an order of magnitude faster than the original architecture in [Mildenhall et al. 2020].

Note that a major focus of our work is using models pretrained on large datasets as-is – to make sure CLIP-Fields can take advantage of the latest advances in the diverse fields it draws from. At the same time, while in our setup we haven't found a need to fine-tune any of the pretrained models mentioned here, we do not believe there is any barrier to do so if such is necessary.

9.4 Approach

In this section, we describe our concrete problem statement, the components of our semantic scene model, and how those components connect with each other.

9.4.1 Problem Statement

We aim to build a system that can connect points of a 3D scene with their visual and semantic meaning. Concretely, we design CLIP-Fields to provide an interface with a pair of scene-dependent implicit functions $f, h : \mathbb{R}^3 \rightarrow \mathbb{R}^n$ such that for the coordinates of any point P in our scene, $f(P)$ is a vector representing its semantic features, and $h(P)$ is another vector representing its visual features. For ease of decoding, we constrain the output spaces of f, h to match the embedding space of pre-trained language and vision-language models, respectively. For the rest of this paper, we refer to such functions as "spatial memory" or "geometric database" since they connect the scene coordinates with scene information.

Given such a pair of functions, we can solve multiple downstream problems in the following way:

- **Segmentation:** For a pixel in a scene, find the corresponding point P_i in space. Use the alignment between a label embedding and $f(P_i)$ to find the label with the highest probability for that pixel. Segment a scene image by doing so for each pixel.
- **Object navigation:** For a given semantic query q_s (or a visual query q_v) find the associated embeddings from our pretrained models, e_s (respectively, e_v), and find the point in space that maximizes $e_s \cdot f(P^*)$ (or $e_v \cdot h(P^*)$). Navigate to P^* using classic navigation stack.
- **View localization:** Given a view v from the scene, find the image embedding e_v of v using the same vision-language model. Find the set of points with highest alignment $e_v \cdot h(P)$ in the scene.

While such a pair of scene-dependent functions f, h would be straightforward to construct if we were given a dataset $\{(P, f(P), h(P) \mid P \in \text{scene}\}$, to make it broadly applicable, we create CLIP-Fields to be able to construct f, h from easily collectable RGB-D videos and odometry data.

9.4.2 Dataset Creation

We assume that we have a series of RGB-D images of a scene alongside odometry information, i.e. the approximate 6D camera poses while capturing the images. As described in 9.5.2, we captured such a dataset using accessible consumer devices such as an iPhone Pro or iPads. To train our model, we first preprocess this set of RGB-D frames into a scene dataset (Fig. 9.2). We convert each of our depth images to pointclouds in world coordinates using the camera's intrinsic and extrinsic matrices. Next, we label each of the points P in the pointcloud with their possible representation vectors, $f(P), h(P)$. When no human annotations are available, we use web-image trained object detection models on our RGB images. We choose Detic [Zhou et al. 2022] as our detection model since it can perform object detection with an open label set. However, this model can freely be swapped out for any other pretrained detection or segmentation model. When available, we can also use human annotations for semantic or instance segmentations.

In both cases, we derive a set of detected objects with language labels in the image, along with their label masks and confidence scores. We back-project the pixels included in the the label mask to the world coordinates using our point cloud. We label each back-projected point in the world with the associated language label and label confidence score. Additionally, we label each back-projected point with the CLIP embedding of the view it was back-projected from as well as the distance between camera and the point in that particular point. Note that each point can appear multiple times in the dataset from different training images.

Thereby, we get a dataset with two sets of labels from our collected RGB-D frames and odometry information. One set of label captures primarily semantic information, $D_{\text{label}} = \{(P, \text{label}_P, \text{conf}_P)\}$

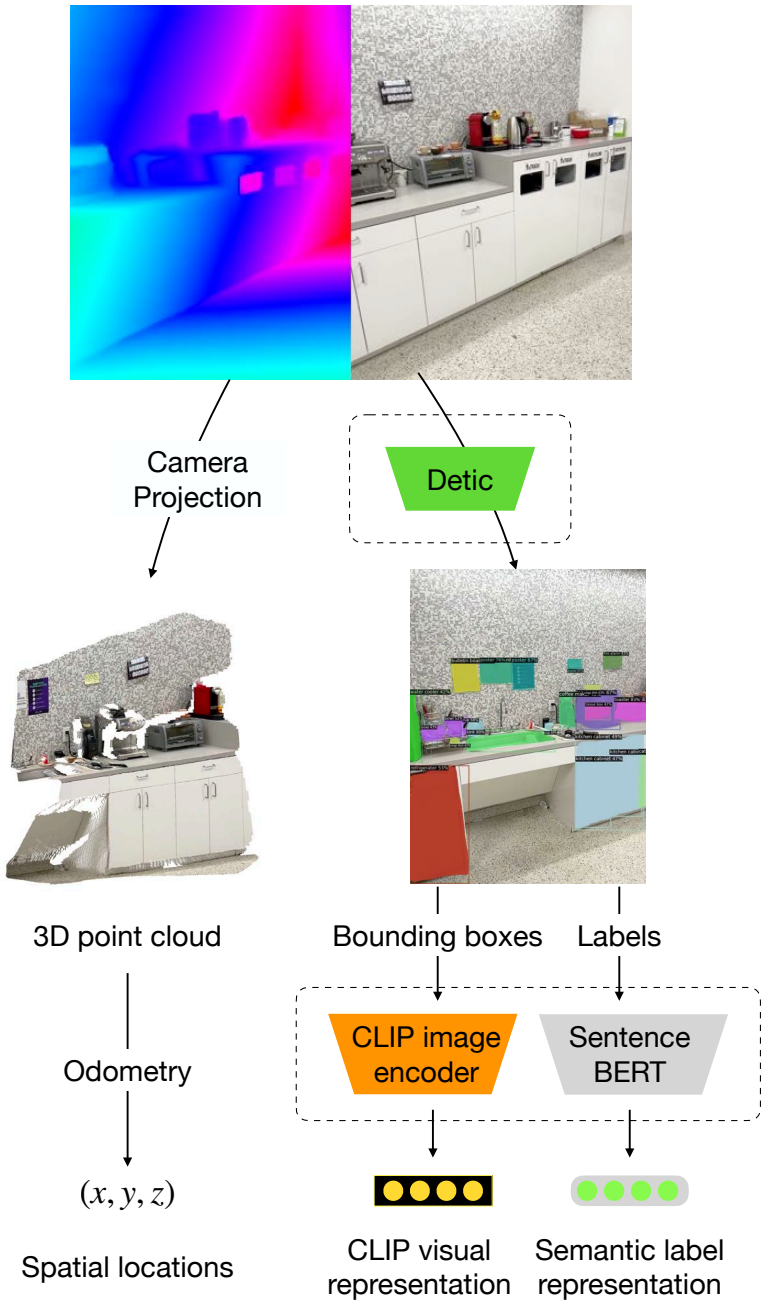


Figure 9.2: Dataset creation process for CLIP-Fields by processing each frame of a collected RGB-D video. Models highlighted by dashed lines are off-the-shelf pre-trained models, showing that we can train a real world CLIP-Fields using no direct human supervision beyond pre-trained open label object detectors, large language models (LLMs) and visual language models (VLMs).

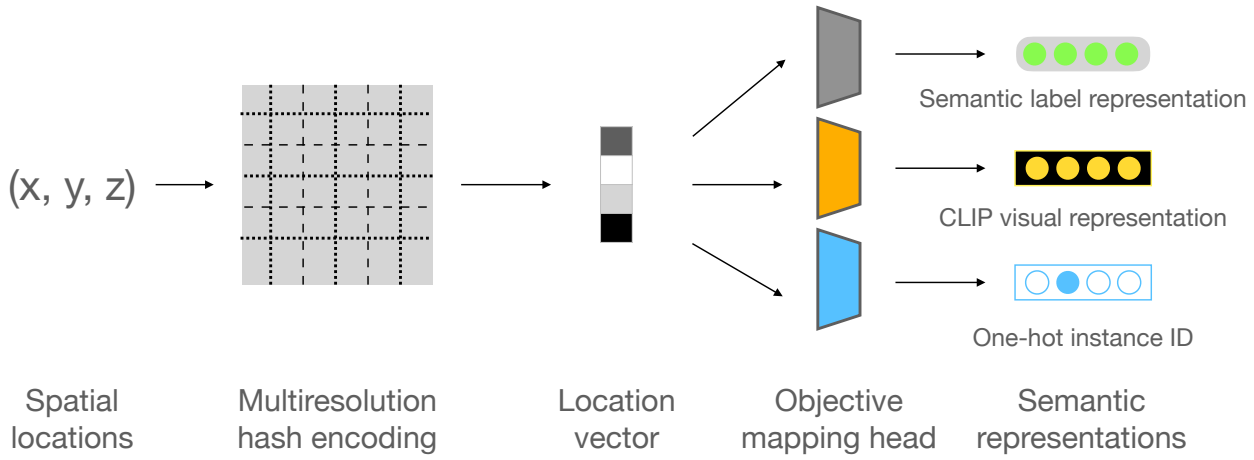


Figure 9.3: Model architecture for CLIP-Fields. We use a Multi-resolution Hash Encoder [Müller et al. 2022] to learn a low level spatial representation mapping $\mathbb{R}^3 \rightarrow \mathbb{R}^d$, which is then mapped to higher dimensions and trained with contrastive objectives.

where label_P and conf_P are just detector-given label and the confidence score to such label for each point. The second set of labels captures primarily visual information, $D_{\text{image}} = \{(P, \text{clip}_P, \text{dist}_P)\}$, where clip_P is the CLIP embedding of the image point P was back-projected from, and dist_P is the distance between P and the camera in that image. We then train CLIP-Fields to efficiently combine the representations, encoding the points' semantic and visual properties in g.

9.4.3 Model Architecture

CLIP-Fields can be divided into two components: a trunk $g : \mathbb{R}^3 \rightarrow \mathbb{R}^d$, which maps each location (x, y, z) to a representation vector, and individual heads, one for each one of our objectives, like language or visual representation retrieval. See Figure 9.3 for an overview.

We can parameterize g with any neural field architecture; in CLIP-Fields we use multi-resolution hash encoding (MHE) as introduced in Instant-NGP [Müller et al. 2022], with $d = 144$. MHEs build an implicit representation over coordinates with a feature-pyramid like structure, which can flexibly maintain both local and global information, unlike purely voxel-based encodings which focuses on local structures only. We primarily use the MHE over other implicit field representations

because we found that they train significantly faster in our datasets. The objective-specific heads are simple two-layer MLPs with ReLU nonlinearities that map the 144 dimensional outputs of g into higher dimensions which depend on the associated objective. These include head_s that outputs a vector that matches a natural language description of what is at the point in space, and head_v that matches the visual appearance of the object occupying that point in space. Optionally, we can include an instance identification head if we have the appropriate labels to train it.

9.4.4 Objectives

The functions f, h in our implicit scene model can be simultaneously trained with multiple objectives. Each objective trains an implicit function that maps from real world locations in $\mathbb{R}^3$ to the objective space. CLIP-Fields are trained on a specific scene with a contrastive loss, similar to CLIP [Radford et al. 2021]. While training the contrastive loss objective, we also take into consideration the associated label weights. For the contrastive loss calculation, the loss is weighted by the label confidence (for semantic labels, like label embeddings from SentenceBERT [Reimers and Gurevych 2019]), or negative exponential of distance from camera to point (for visual labels from CLIP [Radford et al. 2021] embeddings). Additionally, as is standard practice, we scale the dot product of the predicted and the ground truth embeddings by a learned temperature value. We use the following training objectives:

Semantic Label Embedding: This objective trains the function encoding the semantic information of a 3D point as a n-dimensional representation vector. We train this using the assigned natural language labels to each point. We first convert each label to a semantic vector using a pre-trained language model trained to compare semantic similarity, such as CLIP [Radford et al. 2021] or Sentence-BERT [Reimers and Gurevych 2019]. In this paper we used Sentence-BERT for these language features with $n = 768$.

Mathematically, let us assume that P is the point where we are calculating the loss, P^- are points

with a different semantic label, $f = \text{head}_s \circ g$ is the associated semantic encoding function, $\mathcal{F}$ is a pre-trained semantic language encoder, c is the confidence associated with the label at P, and τ is a temperature term, then the semantic label loss is:

$$\text{L}_{\mathcal{L}}(P, f(P)) = -\text{c} \log \frac{\exp\left(f(P)^T \mathcal{F}(\text{label}_P)/\tau\right)}{\sum_{P^-} \exp\left(f(P)^T \mathcal{F}(\text{label}_{P^-})/\tau\right)}$$

Visual Feature Embedding: This objective trains the embedding of the language-aligned visual context of each scene point into a single vector, akin to CLIP [Radford et al. 2021]. We define the visual context of each point as a composite of the CLIP embedding of each RGB frame this point was included in, weighted by the distance from camera to the point in that frame. If it is possible to do so from the given annotation, we limit the image embedding to only encode what is in the associated object's bounding box. Detic [Zhou et al. 2022], for example, produces embeddings for region proposals for each detected objects, which we use. In this paper's experiments, we use the CLIP ViT-B/32 model embeddings, giving the visual features 512 dimensions.

Similar to the previous objective, given CLIP visual embedding Cs associated with the points, the mapping $h = \text{head}_v \circ g$, the distance between camera and the positive point d_P, and temperature term τ, the visual context loss L_C, is:

$$\text{L}_\text{C}(P, h(P)) = -e^{-d_P} \log \frac{\exp\left(h(P)^T \text{C}_P/\tau\right)}{\sum_{P^-} \exp\left(h(P)^T \text{C}_{P^-}/\tau\right)},$$

Auxilary objectives like Instance Identification: This optional head projects the point representation to a one-hot vector identifying its instance. We use this projection head only in the cases where we have human labeled instance identification data from the scene, and the projection dimension is number of identified instances, plus one for unidentified instances. Instance identification one-hot vectors are trained with a simple cross-entropy loss L_I.

Then, the final loss for CLIP-Fields becomes

$$L = \mathrm{L}_{\mathcal{L}} + \mathrm{L}_{\mathrm{C}} + \alpha L_I$$

where α is a normalizing hyper-parameter to bring the cross-entropy loss to a comparable scale of the contrastive losses.

9.4.5 Training

Our models are trained with the datasets described in Sec. 9.4.2. We train the implicit maps simultaneously with the contrastive losses described in Sec. 9.4.4. Under this loss, each embedding is pushed closer to positive labels and further away from negative labels. For the label embedding head, the positive example is the semantic embedding of the label associated with that point, while negative examples are semantic embeddings of any other labels. For the visual context embedding head, the positive examples are the embeddings of all images and image patches that contain the point under consideration, while the negative examples are embeddings of images that do not contain that point. Similar to CLIP [Radford et al. 2021], we also note that a larger batch size helps reduce the variance in the contrastive loss function. We use a batch size of 12, 544 everywhere since that is the maximum batch size we could fit in our VRAM of an NVIDIA Quadro GP100 GPU.

9.5 Experimental Evaluation

We evaluate CLIP-Fields in terms of instance and semantic segmentation in images first – to show that given ground truth data, it can learn meaningful scene representations. Then, we show that, only using weak web-model supervision, CLIP-Fields can be used as a robot's spatial memory with semantic information. Our visual segmentation experiments are performed on a subset of

Habitat-Matterport 3D Semantic (HM3D semantics) [Yadav et al. 2023] dataset, while our robot experiments were performed on a Hello Robot Stretch using Hector SLAM [Kohlbrecher et al. 2011]. We chose HM3D semantics as our sim testing ground because in this dataset, each scene comes with a different set of labels derived from free-form annotations.

9.5.1 Instance and semantic segmentation in scene images

The first task that we evaluate our model on is learning instance and semantic segmentation of 3D environments. We assume that we have access to a scene, a collection of RGB-D images in it from different viewpoints, and a limited number of them are annotated either by humans, or by a model. We consider two cases in this scenario: one where there are some human annotation data available, and in another where we are completely reliant on large, web-image trained models.

Baselines

In our semantic and instance segmentation tasks, we use 2D RGB based segmentation models as our baselines. In all of the few-shot segmentation experiments, we take a Mask-RCNN model with a ResNet50 FPN backbone, and a DeepLabV3 model with a ResNet50 backbone. All baseline models were pre-trained on ImageNet-1K and then the COCO dataset. We fine-tune the final layers of these pretrained models on each of our limited datasets, and then evaluate them on the held-out set. For the RN50 FPN model, we report the mAP at [0.5-0.95] IoU range. Detic is absent from the first two evaluations since it is a detection model and thus cannot be fine-tuned on segmentation labels.

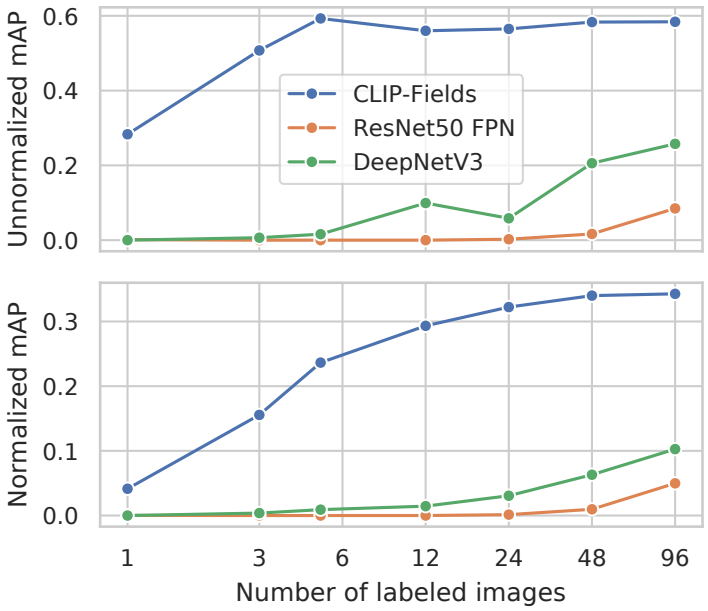


Figure 9.4: Mean average precision in instance segmentation on the Habitat-Matterport 3D (HM3D) Semantic dataset, (top) calculated over only seen instances, and (bottom) calculated over all instances.

Evaluating CLIP-Fields

Since CLIP-Fields defines a function that maps from 3D coordinates, rather than from pixels, to representation vectors, to evaluate this model's learned representations we also have to use the depth and odometry information associated with the image. To get semantic or instance segmentation, we take the depth image, using the camera matrix and odometry project it back to world coordinates, and then query the associated points in world coordinate from CLIP-Fields to retrieve the associated representations with the points. These representations can once again be projected back into the camera frame to reconstruct the segmentation map predicted by CLIP-Fields. Back-projecting to 3D world coordinates also lets CLIP-Fields correctly identify visually occluded and obstructed instances in images, which is not easy for RGB-only models.

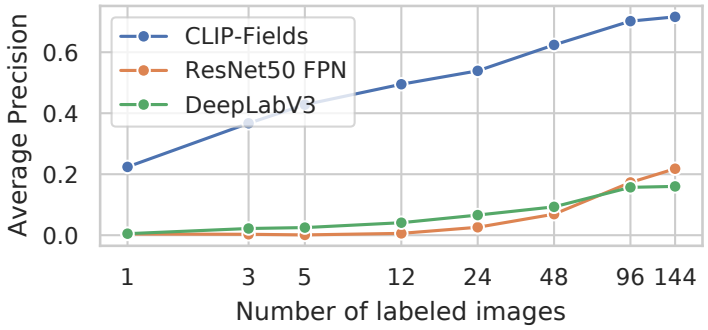


Figure 9.5: Mean average precision in semantic segmentation on the Habitat-Matterport 3D (HM3D) Semantic dataset. Here, the average precision numbers are averaged over all semantic classes.

9.5.1.1 Low-shot instance identification

In this setting, we assume that we have access to a few images densely annotated with an instance segmentation with associated instance IDs. Such annotations are difficult for a human to provide, and thus it is crucial in this setting to perform well with very few (1-5) examples.

On this setting, we train CLIP-Fields with the provided instance segmented RGB-D images and the associated odometry data, and compare with the baseline pretrained 2D RGB models fine-tuned on the same data.

As we can see in Figure 9.4, the average precision of the predictions retrieved from CLIP-Fields largely outperforms the RGB-models. This statement holds true whether we normalize by the number of seen instances in the training set or by the total number of instances in the scene.

9.5.1.2 Low-shot semantic segmentation

Next, we focus on a similar setting on semantically segmenting the views from the scene from a few annotations.

In Figure 9.5, we see once again that CLIP-Fields outperforms the RGB-based models significantly, to the point where even with three labelled views, CLIP-Fields has a higher AP than any of the

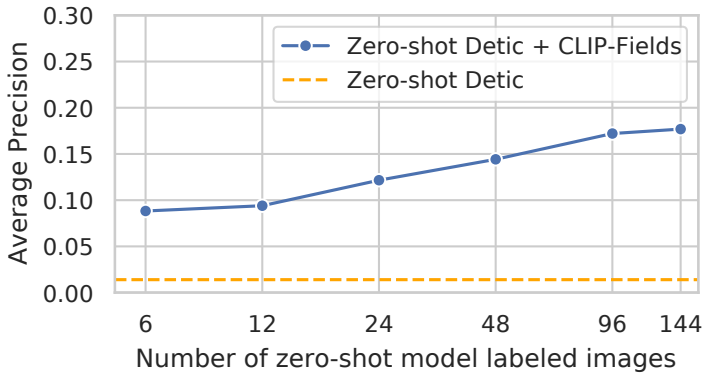


Figure 9.6: Mean average precision in zero-shot semantic segmentation on the Habitat-Matterport 3D (HM3D) Semantic dataset.

baseline RGB models.

9.5.1.3 Zero-shot semantic segmentation

To examine the benefits derived purely from imposing multi-view consistency and a 3D structure over 2D model predictions, we experiment with CLIP-Fields trained solely with labels from large web-image trained models in a zero-shot settings. In this experiment, we train CLIP-Fields only with labels given to us by such large web models, namely Detic [Zhou et al. 2022]. We get the labels by using Detic on the unlabeled training images, and then train CLIP-Fields on it. Besides text labels from Detic, we also use the CLIP visual representations to augment the implicit model, as described in Section 9.4.4.

As a baseline, we compare the trained CLIP-Fields with performance of the same Detic model used to label the scene images. Both CLIP-Fields and the baseline had access to the list of semantic labels in each scene with no extra annotations. We see in Figure 9.6 that enforcing 3D structure and multi-view consistency in our segmentation predictions improves the test-time predictions considerably.

In all our visual segmentation experiments, we see that enforcing 3D consistency and structure

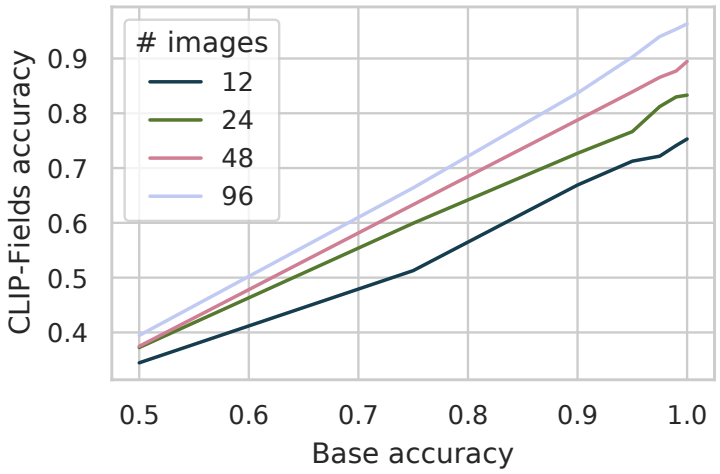


Figure 9.7: Mean average accuracy on the semantic segmentation task on the HM3D Semantic dataset with label noise simulating errors in base labelling models. Different lines show performance of models trained with a different number of labeled training frames.

using CLIP-Fields helps identifying scene properties from images. Back-projecting the rays can also help CLIP-Fields correctly identify objects which are occluded and partially visible. This property can be extremely helpful in a busy indoor setting where not every object can be visible from every angle. Ability to work with occluded views and partial information can be a strong advantage for any embodied intelligent agent.

9.5.1.4 CLIP-Fields's robustness to label errors

In real-world applications, CLIP-Fields relies on labels given by large-scale web-data trained models, which rarely (if ever) have perfect accuracy. In this section, we examine the robustness of CLIP-Fields to such label errors. In this experiment, we simulate label errors by taking ground truth semantic labels in simulation, and for each frame and each object in that frame, flipping that object's label to another random label with probability p. By doing so, we simulate labelling our training data by a model whose mean accuracy is $1 - p$.

We see from Figure 9.7 that as the base model's semantic label prediction accuracy increases,

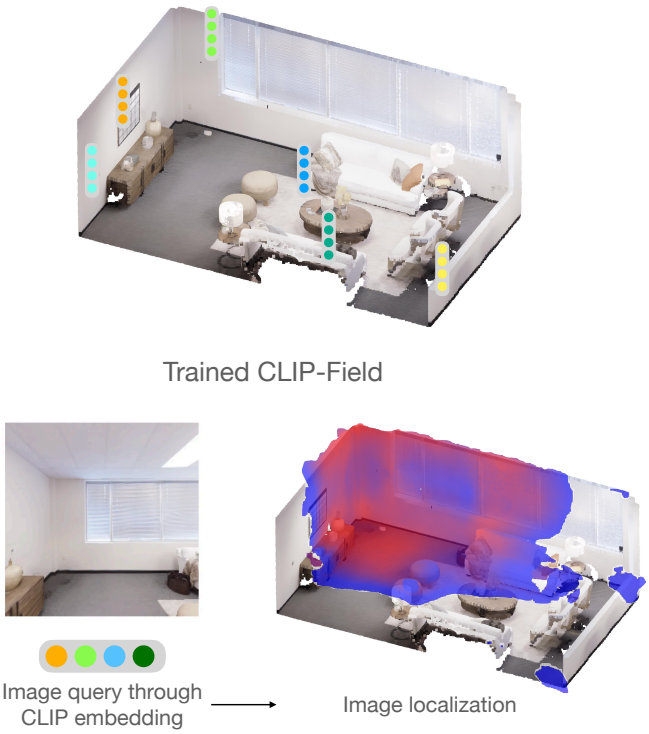


Figure 9.8: View localization using a trained CLIP-Fields. We encode the query image on the bottom left to its CLIP representation, and visualize the locations whose CLIP-Fields representations have the highest (more red) dot product with the embedded image. Lower dot products are blue; and below a threshold are uncolored.

CLIP-Fields's label prediction accuracy increases almost linearly. Importantly, there is no dramatic accuracy decrease when the base model accuracy goes below 1. Thus, we can see that CLIP-Fields maintain reasonable accuracy as long as the base models are also reasonably accurate, which is the case for the state-of-the-art detection and segmentation models. As the base models naturally improve over time with continuous efforts in the computer vision and natural language processing fields, we expect CLIP-Fields's performance to improve correspondingly.

9.5.1.5 View Localization

Since CLIP-Fields is trained with CLIP embeddings at each coordinate, we can use such embeddings to localize an arbitrary view from the scene. To do so, we simply find the CLIP embedding of the

Figure 9.9: Scenes for our real-world semantic navigation experiments. The top scene is a lab kitchen and the bottom is a library/lounge.

query image. Then, we query the visual representation of the points in the scene, and take the dot product between the query representation and the point representations. Due to the contrastive loss that CLIP was trained with, points that have similar embeddings to the query embedding will have the highest dot product. We can use this principle to localize any view in the scene, as shown in Figure 9.8.

9.5.2 Semantic Navigation on Robot with CLIP-Fields as Semantic-Spatial Memory

Training a CLIP-Fields with available data, whether they are labeled by humans or pretrained models, gives us a mapping from real world coordinates to a vector representation trained to

contain their semantic and visual properties (Section 9.4.4). In this section, we evaluate the quality of the learned representations by using the learned model for downstream robot semantic navigation tasks.

9.5.2.1 Task setup

We define our robot task in a 3D environment as a "Go and look at X" task, where X is a natural language query defined by the user. To test CLIP-Fields's semantic understanding capabilities, we formulate the queries from three different categories:

- *Literal queries:* At this level, we choose X to be the literal and unambiguous name of an object present in the scene, such as "the refrigerator" or "the typewriter".
- *Visual queries:* At this level, we add references to objects by their visual properties, such as "the red fruit bowl" or "the blue book with a house on the cover".
- *Semantic queries:* At this level, we add references to objects by their semantic properties, such as "warm my lunch" (microwave), or "something to read" (a book).

9.5.2.2 Data collection and training

We ran our robot experiment in two different scenes, one in the lab kitchen, and another in the lab library (Figure 9.9). For each of the scenes, we collected the RGB-D and odometry data with an iPhone 13 Pro with LiDAR sensors. The iPhone recording gave us a sequence of RGB-D images as well as the approximate camera poses in real world coordinate. On each of these scenes, we labelled a subset of the collected RGB images with Detic [Zhou et al. 2022] model using ScanNet200 [Rozenberszki et al. 2022] labels. Then, we created a training dataset with 3D world coordinates and their associated semantic and visual embeddings using the method described

Figure 9.10: Examples of the robot's semantic navigation in two different testing environments, looking at objects given different queries. The images show the robot's POV given the associated query, with color coded borders showing approximate correctness. The rows show different two different scenes, top being in a lab kitchen and the bottom in our lab's library/lounge space, shown in detail in figure 9.9.

in Section 9.4.2. On this dataset, we trained a CLIP-Fields to synthesize all the views and their associated labels.

9.5.2.3 Robot execution

Next, on our robot, we load the CLIP-Field to help with the localization and navigation of the robot. When the robot gets a new text query, we first convert it to a representation vector. We use Sentence-BERT to retrieve the semantic part of the query representation and CLIP text model to retrieve the vision-aligned part of the query representation. Then, we find the coordinates of the point P in space that has the highest alignment with the query representations, as described in Section 9.4.1 and Figure 9.11. We use the robot's Hector SLAM [Kohlbrecher et al. 2011] navigation stack to navigate to that region, and point the robot camera to an XYZ coordinate where the dot product was highest. We consider the navigation task successful if the robot can navigate to and point the camera at an object that satisfies the query. We run twenty queries in the kitchen and fifteen queries in the library environment.

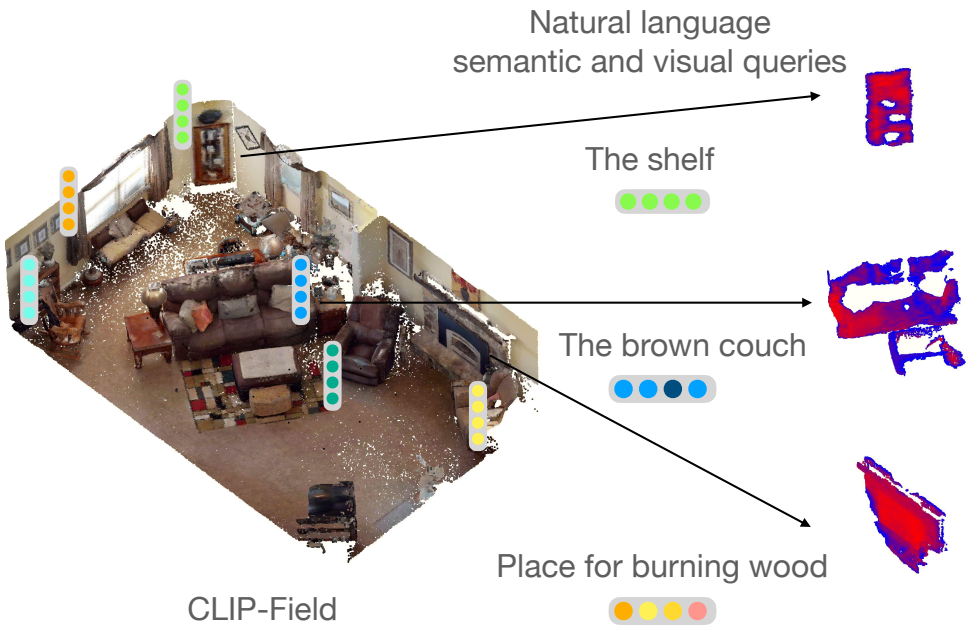


Figure 9.11: Running semantic queries against a trained CLIP-Fields. We encode our queries with language encoders, and compare the encoded representation with the stored representation in CLIP-Fields to then extract the best matches.

9.5.2.4 Experiment results

In our experiments (Figure 9.10), we see that CLIP-Fields let the robot navigate to different points in the environment from semantic natural language queries. We generally observe that if an object was correctly identified by the web-image models during data preparation, when queried literally CLIP-Fields can easily understand and navigate to it, even with intentional misspellings in the query. However, if an object was misidentified during data preparation, CLIP-Fields fails to correctly identify it as well. For example, in row two, column two of Figure 9.10, the part of the floor that is identified as a “table" was identified as a “table" by our web-image model earlier. This observation lines up with our simulated experiments in Section 9.5.1.4 where we saw that CLIP-Fields performance has a linear relationship with the base models’ performance. For semantic queries, CLIP-Fields sometimes confuses two related concepts; for example, it retrieves the dishwasher for both “place to wash my hand" and “place to wash my dishes". Finally, the visual

queries sometimes put a higher weight on the semantic match rather than visual match, such as retrieving a white fruit bowl for "red fruit bowl" instead of the red bowl in the scene. However, the right object is retrieved if we query for "red plastic bowl".

We have presented detailed logs of running CLIP-Fields on the robot in the kitchen environment in Appendix F.2 detailing all the queries and the resulting robot behavior.

9.6 Limitations

We showed that CLIP-Fields can learn 3D semantic scene representations from little or no labeled data, relying on weakly-supervised web-data trained models, and that we can use these model to perform a simple “look-at” task in the real world. CLIP-Fields allow us to answer queries of varying levels of complexity. We expect this kind of 3D representation to be generally useful for robotics. For example, it may be enriched with affordances for planning; the geometric database can be combined with end-to-end differentiable planners. In future work, we hope to explore models that share parameters across scenes, and can handle dynamic scenes and objects.

Postscript

With a few year’s worth of hindsight, it is amazing the kind of new capabilities that has been unlocked in the near term by spatio-semantic memories – including the ability to chain small-spatial scale policies into mobile manipulation systems. However, the limitation of CLIP-fields was in its implicit memory representation – particularly, that neural fields were not the best representations for a system that needs to perform frequent, small scale local updates to the memory. These practical considerations were answered in later iterations in Chapter 10 and Chapter 11. Finally, we need to understand and resolve the balance between implicit and explicit memory going forward.

Acknowledgements

This work was co-authored with Chris Paxton, Lerrel Pinto, Soumith Chintala, and advised by Arthur Szlam during an internship at Meta Fundamental AI Research (FAIR) in 2022.

10 Integrating Open-knowledge Models for Robotics: OK-Robot

10.1 Introduction

Creating a general-purpose robot has been a longstanding dream of the robotics community. With the increase in data-driven approaches and large robot models, impressive progress is being made [Pinto and Gupta 2016; Levine et al. 2018; Brohan et al. 2023b; Shafiullah et al. 2023b]. However, current systems are brittle, closed, and fail when encountering unseen scenarios. Even the largest robotics models can often only be deployed in previously seen environments [Brohan et al. 2023a; Zitkovich et al. 2023]. The brittleness of these systems is further exacerbated in settings where little robotic data is available, such as in unstructured home environments.

The poor generalization of robotic systems lies in stark contrast to large vision models [Zhou et al. 2022; Minderer et al. 2022; Radford et al. 2021; Marino et al. 2019], which show capabilities of semantic understanding [Alayrac et al. 2022; Liu et al. 2023b,a], detection [Zhou et al. 2022; Minderer et al. 2022], and connecting visual representations to language [Radford et al. 2019, 2021; Marino et al. 2019] At the same time, base robotic skills for navigation [Gervet et al. 2023a], grasping [Sundermeyer et al. 2021; Mahler et al. 2017b; Fang et al. 2020, 2023c], and rearrangement [Goyal et al. 2022b; Liu et al. 2022] are fairly mature. Hence, it is perplexing that robotic

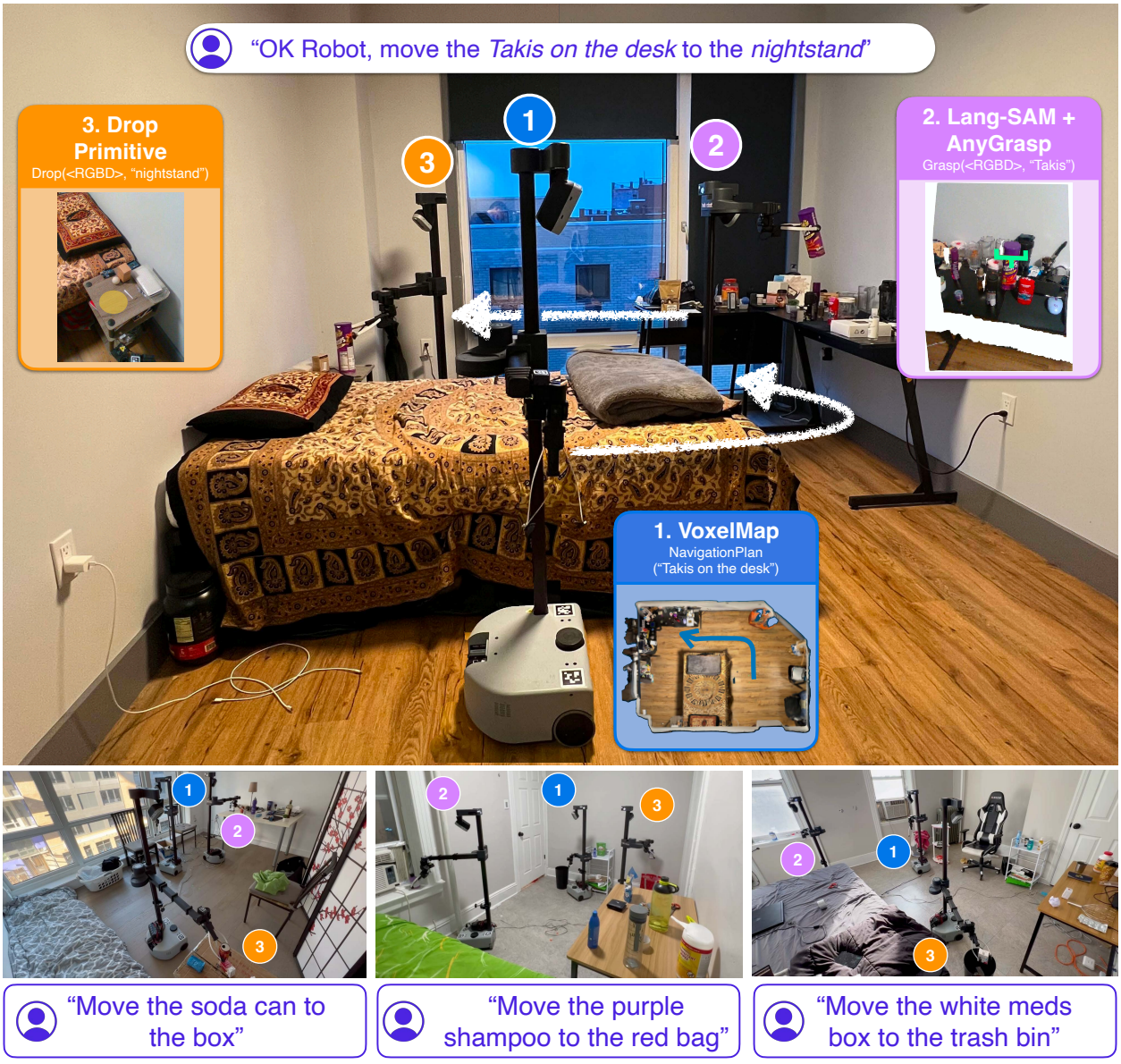


Figure 10.1: OK-Robot is an Open Knowledge robotic system, which integrates a variety of learned models trained on publicly available data, to pick and drop objects in real-world environments. Using Open Knowledge models such as CLIP, Lang-SAM, AnyGrasp, and OWL-ViT, OK-Robot achieves a 58.5% success rate across 10 unseen, cluttered home environments, and 82.4% on cleaner, decluttered environments.

systems that combine modern vision models with robot-specific primitives perform so poorly. To highlight the difficulty of this problem, the recent NeurIPS 2023 challenge for open-vocabulary mobile manipulation (OVMM) [Yenamandra et al. 2023a] registered a success rate of 33% for the winning solution [Melnik et al. 2023].

So what makes open-vocabulary robotics so hard? Unfortunately, there isn't a single challenge that makes this problem hard. Instead, inaccuracies in different components compound and together results in an overall drop. For example, the quality of open-vocabulary retrievals of objects in homes is dependent on the quality of query strings, navigation targets determined by VLMs may not be reachable to the robot, and the choice of different grasping models may lead to large differences in grasping performance. Hence, making progress on this problem requires a careful and nuanced framework that both integrates VLMs and robotics primitives, while being flexible enough to incorporate newer models as they are developed by the VLM and robotics community.

We present `OK-Robot`, an *Open Knowledge Robot* that integrates state-of-the-art VLMs with powerful robotics primitives for navigation and grasping to enable pick-and-drop. Here, *Open Knowledge* refers to learned models trained on large, publicly available datasets. When placed in a new home environment, `OK-Robot` is seeded with a scan taken from an iPhone. Given this scan, dense vision-language representations are computed using LangSam [Medeiros 2023] and CLIP [Radford et al. 2021] and stored in a semantic memory. Then, when a language-query for an object to be picked comes in, semantic memory is queried with the language embedding to find that object. After this, navigation and picking primitives are applied sequentially to move to the desired object and pick it up. A similar process can be carried out for dropping the object.

To study `OK-Robot`, we tested it in 10 real world home environments. Through our experiments, we found that on a unseen natural home environment, a zero-shot deployment of our system achieves 58.5% success on average. However, this success rate is largely dependant on the "naturalness" of the environment, as we show that with improving the queries, decluttering the space, and

excluding objects that are clearly adversarial (too large, too translucent, too slippery), this success rate reaches 82.4%. Overall, through our experiments, we make the following observations:

- **Pre-trained VLMs are highly effective for open-vocabulary navigation:** Current open-vocabulary vision-language models such as CLIP [Radford et al. 2021] or OWL-ViT [Minderer et al. 2022] offer strong performance in identifing arbitrary objects in the real world, and enable navigating to them in a zero-shot manner (see Section 10.2.1.)
- **Pre-trained grasping models can be directly applied to mobile manipulation:** Similar to VLMs, special purpose robot models pre-trained on large amounts of data can be applied out of the box to approach open-vocabulary grasping in homes. These robot models do not require any additional training or fine-tuning (see Section 10.2.2.)
- **How components are combined is crucial:** Given the pretrained models, we find that they can be combined with no training using a simple state-machine model. We also find that using heuristics to counteract the robot's physical limitations can lead to a better success rate in the real world (see Section 10.2.4.)
- **Several challenges still remain:** While, given the immense challenge of operating zero-shot in arbitrary homes, `OK-Robot` improves upon prior work, by analyzing the failure modes we find that there are significant improvements that can be made on the VLMs, robot models, and robot morphology, that will directly increase performance of open-knowledge manipulation agents (see Section 10.3.4).

To encourage and support future work in open-knowledge robotics, we will share the code and modules for `OK-Robot`, and are committed to supporting reproduction of our results. More information along with robot videos are available on our project website: `https://ok-robot.github.io/`.

(a) Open-vocabulary object localization using VoxelMap

(b) Open-vocabulary navigation planning using VoxelMap and heuristics weighted A*

Figure 10.2: Open-vocabulary, open knowledge object localization and navigation in the real-world. We use the VoxelMap [Yenamandra et al. 2023b] for localizing objects with natural language queries, and use an A* algorithm similar to USANet [Bolte et al. 2023] for path planning.

10.2 Technical Components and Method

Our method, on a high level, solves the problem described by the query: "Pick up **A** (from **B**) and drop it on/in **C**", where A is an object and B and C are places in a real-world environment such as homes. The system we introduce is a combination of three primary subsystems combined on a Hello Robot: Stretch. Namely, these are the open-vocabulary object navigation module, the open-vocabulary RGB-D grasping module, and the dropping heuristic. In this section, we describe each of these components in more details.

10.2.1 Open-home, open-vocabulary object navigation

The first component of our method is an open-home, open-vocabulary object navigation model that we use to map a home and subsequently navigate to any object of interest designated by a natural language query.

10.2.1.1 Scanning the home

For open vocabulary object navigation, we follow the approach from CLIP-Fields [Shafiullah et al. 2023a] and assume a pre-mapping phase where the home is "scanned" manually using an iPhone. This manual scan simply consists of taking a video of the home using the Record3D app on the iPhone, which results in a sequence of posed RGB-D images and takes less than one minute for a new room. Once collected, the RGB-D images, along with the camera pose and positions, are exported to our library for map-building. To ensure our semantic memory contains both the objects of interest as well as the navigable surface and any obstacles, we capture the floor surface alongside the objects and receptacles in the environment.

10.2.1.2 Detecting objects

On each frame of the scan, we run an open-vocabulary object detector. We chose OWL-ViT [Minderer et al. 2022] over Detic [Zhou et al. 2022] as the object detector since we found OWL-ViT to perform better in preliminary queries. We apply the detector on every frame, and extract each of the object bounding box, CLIP-embedding, detector confidence, and pass these information onto the object memory module. We further refine the bounding boxes into object masks with Segment Anything (SAM) [Kirillov et al. 2023]. Note that, in many cases, open-vocabulary object detectors require a set of natural language object queries to be detected. We supply a large set of such object queries, derived from the original Scannet200 labels [Rozenberszki et al. 2022] and presented in Appendix G.2, to help the detector captures most common objects in the scene.

10.2.1.3 Object-centric semantic memory

We use an object-centric memory similar to Clip-Fields [Shafiullah et al. 2023a] and OVMM [Yenamandra et al. 2023b] that we call the VoxelMap. VoxelMap is built by back-projecting the object

masks in real-world coordinates using the depth image and the pose collected by the camera. This process giving us a point cloud where each point has an associated CLIP semantic vector. Then, we voxelize the point cloud to a 5 cm resolution. For each voxel, we calculate the detector-confidence weighted average for the CLIP embeddings that belong to that voxel. This VoxelMap builds the base of our object memory module. Note that the representation created this way remains static after the first scan, and cannot be adapted during the robot's operation. This inability to dynamically create a map is discussed in our limitations section (Section 10.5).

10.2.1.4 Querying the memory module

Our semantic object memory gives us a static world representation represented as possibly non-empty voxels in the world, and a semantic vector in CLIP space associated with each voxel. Given a language query, we first convert it to a semantic vector using the CLIP language encoder. Then, we find the voxel where the dot product between the encoded embedding and the voxel's associated embedding is maximized. Since each voxel is associated with a real location in the home, this lets us find the location where a queried object is most likely to be found, similar to Figure 10.2(a).

We also implement querying for "A on B" by interpreting it as "A near B". We do so by selecting top-10 points for query A and top-50 points for query B. Then, we calculate the 10×50 pairwise L_2 distances and pick the A-point associated with the shortest (A, B) distance. Note that during the object navigation phase we use this query only to navigate to the object approximately, and not for manipulation. This approach gives us two advantages: our map can be as lower resolution than those in prior work [Shafiullah et al. 2023a; Bolte et al. 2023; Kerr et al. 2023], and we can deal with small movements in object's location after building the map.

10.2.1.5 Navigating to objects in the real world

Once our navigation model gives us a 3D location coordinate in the real world, we use that as a navigation target for our robot to initialize our manipulation phase. Going and looking at an object [Shafiullah et al. 2023a; Gervet et al. 2023a; Chang et al. 2023] can be done while remaining at a safe distance from the object itself. In contrast, our navigation module must place the robot at an arms length so that the robot can manipulate the target object afterwards. Thus, our navigation method has to balance the following objectives:

1. The robot needs to be close enough to the object to manipulate it,
2. The robot needs some space to move its gripper, so there needs to be a small but non-negligible space between the robot and the object, and,
3. The robot needs to avoid collision during manipulation, and thus needs to keep its distance from all obstacles.

We use three different navigation score functions, each associated with one of the above points, and evaluate them on each point of the space to find the best position to place the robot.

Let a random point be $\overrightarrow{x}$, the closest obstacle point as $\overrightarrow{x}_{obs}$, and the target object as $\overrightarrow{x_o}$. We define the following three functions s_1, s_2, s_3 to capture our three criterion. We define s as their weighted sum. The ideal navigation point $\overrightarrow{x}^*$ is the point in space that minimizes $s(\overrightarrow{x})$, and the ideal

direction is given by the vector from $\overrightarrow{x^*}$ to $\overrightarrow{x_o}$.

$$
\begin{aligned}
s_1(\overrightarrow{x}) &= ||\overrightarrow{x} - \overrightarrow{x_o}|| \\
s_2(\overrightarrow{x}) &= 40 - \min(||\overrightarrow{x} - \overrightarrow{x_o}||, 40) \\
s_3(\overrightarrow{x}) &= \begin{cases} 1/||\overrightarrow{x} - \overrightarrow{x}_{obs}||, & \text{if } ||\overrightarrow{x} - \overrightarrow{x}_{obs}||_0 \leq 30 \\ 0, & \text{otherwise} \end{cases} \\
s(\overrightarrow{x}) &= s_1(\overrightarrow{x}) + 8s_2(\overrightarrow{x}) + 8s_3(\overrightarrow{x})
\end{aligned}
$$

To navigate to this target point safely from any other point in space, we follow a similar approach to [Bolte et al. 2023; Huang et al. 2023a] by building an obstacle map from our captured posed RGB-D images. We build a 2D, 10cm×10cm grid of obstacles over which we navigate using the A* algorithm. To convert our VoxelMap to an obstacle map, we first set a floor and ceiling height. Presence of occupied voxels in between them implies the grid cell is occupied, while presence of neither ceiling nor floor voxels mean that the grid cell is unexplored. We mark both occupied or unexplored cells as not navigable. Around each occupied point, we mark any point within a 20 cm radius as also non-navigable to account for the robot's radius and a turn radius. During A* search, we use the s_3 as a heuristic function on the node costs to navigate further away from any obstacles, which makes our generated paths similar to ideal Voronoi paths [Garrido et al. 2006] in our experiments.

10.2.2 Open-vocabulary grasping in the real world

Grasping or physically interacting with arbitrary objects in the real world is much more complex than open-vocabulary navigation. We opt for using a pre-trained grasping model to generate grasp poses in the real world, and filter them with language-conditioning using a modern VLM.

Figure 10.3: Open-vocabulary grasping in the real world. From left to right, we show the (a) robot POV image, (b) all suggested grasps from AnyGrasp [Fang et al. 2023c], (c) object mask given label from LangSam [Medeiros 2023], (d) grasp points filtered by the mask, and (e) grasp chosen for execution.

10.2.2.1 Grasp perception

Once the robot reaches the object location using the navigation method outlined in Section 10.2.1, we use a pre-trained grasping model, AnyGrasp [Fang et al. 2023c], to generate a grasp for the robot. We point the robot's RGB-D head camera towards the object's 3D location, given to us by the semantic memory, and capture an RGB-D image from it (Figure 10.3, column 1). We backproject and convert the depth image to a pointcloud and pass this information to the grasp generation model. Our grasp generation model, AnyGrasp, generates all collision free grasps (Figure 10.3 column 2) for a parallel jaw gripper in a scene given a single RGB image and a pointcloud. AnyGrasp provides us with grasp point, width, height, depth, and a "graspness score", indicating uncalibrated model confidence in each grasp.

10.2.2.2 Filtering grasps using language queries

Once we get all proposed grasps from AnyGrasp, we filter them using LangSam [Medeiros 2023]. LangSam [Medeiros 2023] segments the captured image and finds the desired object mask with a language query (Figure 10.3 column 3). We project all the proposed grasp points onto the image and find the grasps that fall into the object mask (Figure 10.3 column 4). We pick the best grasp using a heuristic. Given a grasp score $\mathcal{S}$ and the angle between the grasp normal and floor normal θ, the new heuristic score is $\mathcal{S} - ({}^{\theta^4}/_{10})$. This heuristic balances high graspness scores with finding flat, horizontal grasps. We prefer horizontal grasps because they are robust to small calibration errors on the robot, while vertical grasps needs better hand-eye calibration to be successful. Robustness to hand-eye calibration errors lead to higher success as we transport the robot to different homes during our experiments.

10.2.2.3 Grasp execution

Once we identify the best grasp (Figure 10.3 column 5), we use a simple pre-grasp approach [Dasari et al. 2023] to grasp our intended object. If $\overrightarrow{p}$ is the grasp point and $\overrightarrow{a}$ is the approach vector given by the grasping model, our robot gripper follows the following trajectory:

$$\langle \overrightarrow{p} - 0.2\overrightarrow{a},\ \overrightarrow{p} - 0.08\overrightarrow{a},\ \overrightarrow{p} - 0.04\overrightarrow{a},\ \overrightarrow{p} \rangle$$

Put simply, our method approaches the object from a pre-grasp position in a line with progressively smaller motions. Moving slower as we approach the object helps the robot not knock over light objects. Once we reach the predicted grasp point, we close the gripper in a close loop fashion to get a solid grip on the object without crushing it. After grasping the object, we lift up the robot arm, retract it fully, and rotate the wrist to have the object tucked over the body. This behavior maintains the robot footprint while ensuring the object is held securely by the robot and doesn't

fall while navigating to the drop location.

10.2.3 Dropping heuristic

After picking up an object, we find and navigatte to the drop location using the same methods described in Section 10.2.1. Unlike in HomeRobot's baseline implementation [Yenamandra et al. 2023b] that assumes that the drop-off location is a flat surface, we extend our heuristic to cover concave objects such as sink, bins, boxes, and bags. First, we segment the point cloud P captured by the robot's head camera using LangSam [Medeiros 2023] similar to Section 10.2.2.2 using the drop language query. Then, we align that segmented point cloud such that X-axis is aligned with the way the robot is facing, Y-axis is to its left and right, and the Z-axis of the point cloud is aligned with the floor normal. Then, we normalize the point cloud so that the robot's (x, y) coordinate is $(0, 0)$, and the floor plane is at $z = 0$. We call this pointcloud P_a. On the aligned, segmented point cloud, we consider the (x, y) coordinates for each point, and find the median values x_m and y_m on each axis. Finally, we find a drop height using $z_{\max} = 0.2 + \max\{z \mid (x, y, z) \in P_a; 0 \leqslant x \leqslant x_m; |y - y_m| < 0.1\}$ on the segmented, aligned pointcloud. We add a small buffer of 0.2 to the height to avoid collisions between the robot and the drop location. Finally, we move the robot gripper above the drop point, and open the gripper to drop the object. While this heuristic doesn't explicitly reason about clutter, in our experiments it performs well on average.

10.2.4 Deployment in homes

Our navigation, pick, and drop primitives are combined to create our robot method that can be applied in any novel home. For a new home environment, we "scan" the room in under a minute. Then, it takes less than five minutes to process the scan into our VoxelMap. Once that is done, the robot can be immediately placed at the base and start operating. From arriving into a completely novel environment to start operating autonomously in it, our system takes under 10 minutes on

average to complete the first pick-and-drop task.

10.2.4.1 Transitioning between modules

The transition between different modules is predefined and happens automatically once a user specifies the object to pick and where to drop it. Since we do not implement error detection or correction, our state machine model is a simple linear chain of steps leading from navigating to object, to grasping, to navigating to goal, and to dropping the object at the goal to finish the task.

10.2.4.2 Protocol for home experiments

To run our experiment in a novel home, we move the robot to a previously unobserved room. We record the scene and create our VoxelMap. Concurrently, we pick between 10-20 objects arbitrarily in each scene that can fit in the robot gripper. These are objects found in the scene, and are not chosen ahead of time. We come up with a language query for each chosen object using GPT-4V [OpenAI 2023] to keep the queries consistent and free of experimenter bias. We query our navigation module to filter out all the navigation failures; i.e. objects that our semantic memory module could not locate properly. Then, we execute pick-and-drop on remaining objects sequentially without resets between trials.

10.3 Experiments

We evaluate our method in two set of experiments. On the first set of experiments, we evaluate between multiple alternatives for each of our navigation and manipulation modules. These experiments give us insights about which modules to use and evaluate in a home environment as a part of our method. On the next set of experiments, we took our robots to 10 homes and ran 171 pick-and-drop experiments to empirically evaluate how our method performs in completely novel

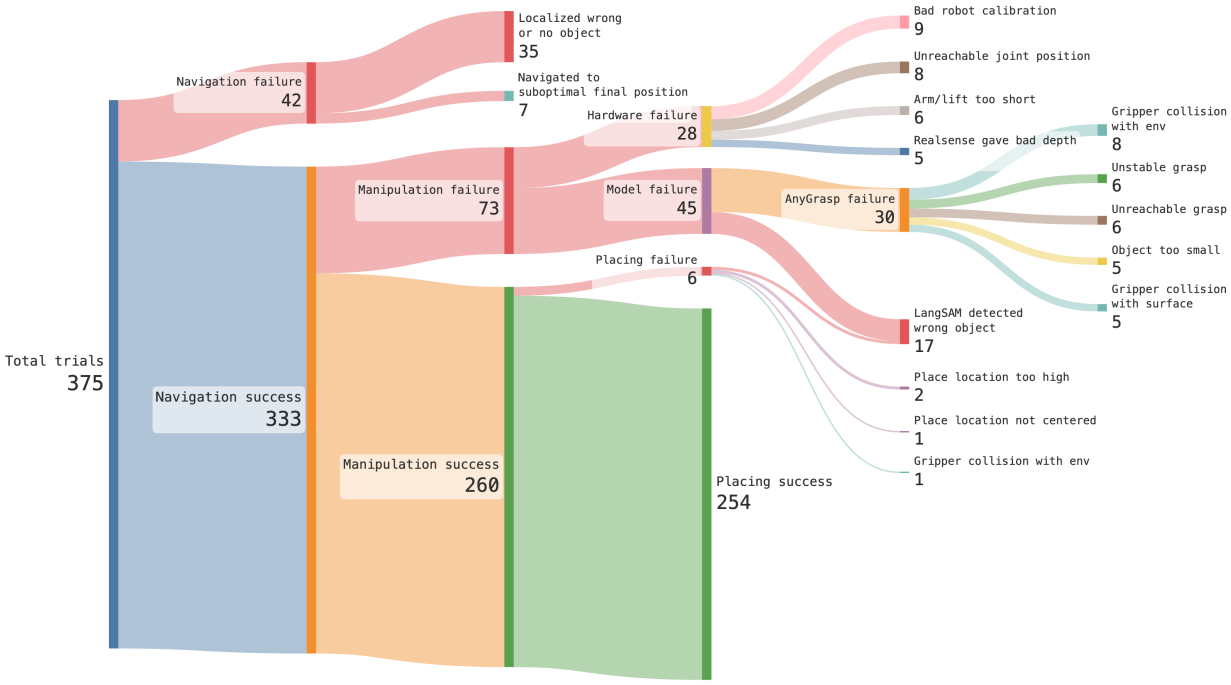


Figure 10.4: All the success and failure cases in our home experiments, aggregated over all three cleaning phases, and broken down by mode of failure. From left to right, we show the application of the three components of OK-Robot, and show a breakdown of the long-tail failure modes of each of the components.

homes, and to understand the failure modes of our system.

Through these experiments, we look to answer a series of questions regarding the capabilities and limits of current Open Knowledge robotic systems, as embodied by OK-Robot. Namely, we ask the following:

1. How well can such a system tackle the challenge of pick and drop in arbitrary homes?
2. How well do alternate primitives for navigation and grasping compare to the recipe presented here for building an Open Knowledge robotic system?
3. How well can our current systems handle unique challenges that make homes particularly difficult, such as clutter, ambiguity, and affordance challenges?
4. What are the failure modes of such a system and its individual components in real home

environments?

10.3.1 Results of home experiments

Over the 10 home environment, OK-Robot achieved a 58.5% success rates in completing full pick-and-drops. Notably, this success rate is over novel objects sourced from each home with our zero-shot algorithm. As a result, each success and failure of the robot tells us something interesting about applying open-knowledge models in robotics, which we analyze over the next sections. In Appendix G.5, we provide the details of all our home experiments and results from the same. In Appendix G.3 we show a subset of the target objects and in Appendix G.4 we show snapshots of homes where OK-Robot was deployed. Snippets of our experiments are in Figure 10.1, and full videos are presented on our project website.

10.3.1.1 Reproduction of our system

Beyond the home experiment results presented here, we also reproduced OK-Robot in two homes in Pittsburgh, PA, and Fremont, CA. These homes were larger and more complex: a cluttered, actively-used home kitchen environment, and a large, controlled test apartment used in prior work [Yenamandra et al. 2023b,a]. In Appendix Figure G.4, we show the robot performing pick-and-drop in these two environments. These homes were different from our initial ten experiments in a few ways. Both were larger compared to the average NY homes, requiring more robot motion to navigate to different goals. The PA environment (Figure G.4 top) notably had much more clutter. However, given only a scan, OK-Robot was able to successfully pick and drop objects like stuffed lion, plush cactus, toy drill, or green water bottle in both environments.

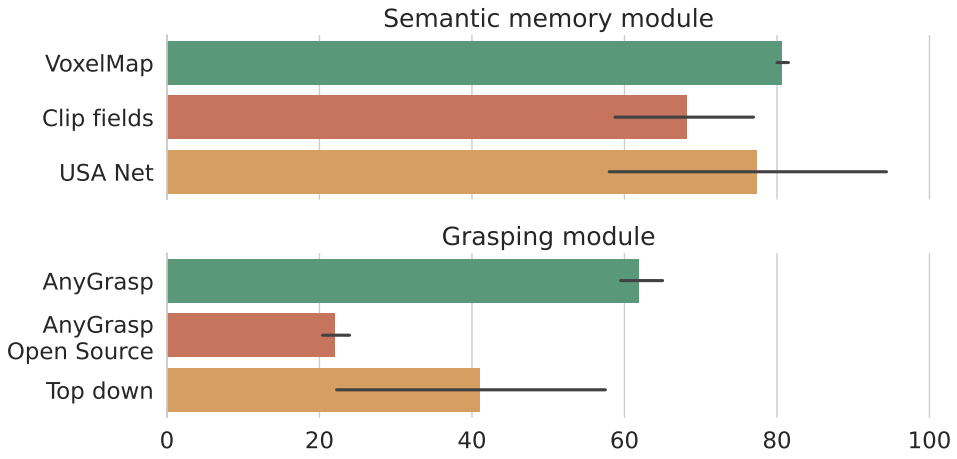


Figure 10.5: Ablation experiment using different semantic memory and grasping modules, with the bars showing average performance and the error bars showing standard deviation over the environments.

10.3.2 Ablations over system components

Apart from the navigation and manipulation strategies used in `OK-Robot`, we also evaluated a number of alternative open vocabulary navigation and grasping modules. We compared them by evaluating them in three different environments in our lab. Apart from VoxelMap [Yenamandra et al. 2023b], we evaluate CLIP-Fields [Shafiullah et al. 2023a], and USA-Net [Bolte et al. 2023] for semantic navigation. For grasping module, we consider AnyGrasp and its open-source baseline, Open Graspness [Fang et al. 2023c], Contact-GraspNet [Sundermeyer et al. 2021], and Top-Down grasp heuristic from home-robot [Yenamandra et al. 2023b]. More details about them are provided in Appendix Section G.1.

In Figure 10.5, we see their comparative performance in three lab environments. For semantic memory modules, we see that VoxelMap, used in `OK-Robot` and described in Sec. 10.2.1.3, outperforms other semantic memory modules by a small margin. It also has much lower variance compared to the alternatives, meaning it is more reliable. As for grasping modules, AnyGrasp clearly outperforms other grasping methods, performing almost 50% better in a relative scale over the next best candidate, top-down grasp. However, the fact that a heuristic-based algorithm,

top-down grasp from HomeRobot [Yenamandra et al. 2023b] beats the open-source AnyGrasp baseline and Contact-GraspNet shows that building a truly general-purpose grasping model remains difficult.

10.3.3 Impact of clutter, object ambiguity, and affordance

What makes home environments especially difficult compared to lab experiments is the presence of physical clutter, language-to-object mapping ambiguity, and hard-to-reach positions. To gain a clear understanding of how such factors play into our experiments, we go through two "clean-up" processes in each environment. During the clean-up, we pick a subset of objects that are free from ambiguity from the previous rounds, clean the clutter around objects, and generally relocated them in an accessible locations. The two clean-up rounds at each environment gives us insights about the performance gap caused by the natural difficulties of a home-like environment.

We show a complete analysis of the tasks listed section 10.3.1 which failed in various stages in Figure 10.6. As we can see from this breakdown, as we clean up the environment and remove the ambiguous objects, the navigation accuracy goes up, and the total error rate goes down from 15% to 12% and finally all the way down to 4%. Similarly, as we clean up clutters from the environment, we find that the manipulation accuracy also improves and the error rates decrease from 25% to 16% and finally 13%. Finally, since the drop-module is agnostic of the label ambiguity or manipulation difficulty arising from clutter, the failure rate of the dropping primitive stays roughly constant through the three phases of cleanup.

10.3.4 Understanding the performance of OK-Robot

While our method can show zero-shot generalization in completely new environments, we probe OK-Robot to better understand its failure modes. Primarily, we elaborate on how our model

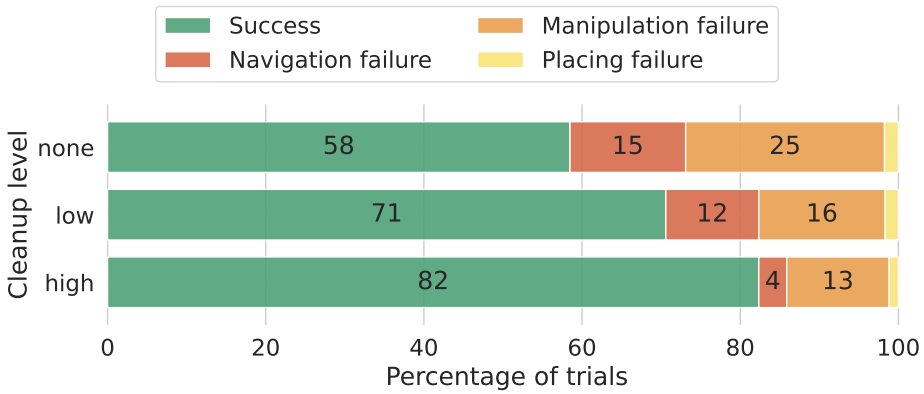


Figure 10.6: Failure modes of our method in novel homes, broken down by the failures of the three modules and the cleanup levels.

performed in novel homes, what were the biggest challenges, and discuss potential solutions to them.

We first show a coarse-level breakdown of the failures, only considering the three high level modules of our method in Figure 10.6. We see that generally, the leading cause of failure is our manipulation failure, which intuitively is the most difficult as well. However, at a closer look, we notice a long tail of failure causes presented in figure 10.4.

The three leading causes of failures are failing to retrieve the right object to navigate to from the semantic memory (9.3%), getting a difficult pose from the manipulation module (8.0%), and robot hardware difficulties (7.5%). In this section, we go over the analysis of the failure modes presented in Figure 10.4 and discuss the most frequent cases.

10.3.4.1 Natural language queries for objects

One of the primary reasons our `OK-Robot` can fail is when a natural language query given by the user doesn't retrieve the intended object from the semantic memory. In Figure 10.7 we show how some queries may fail while semantically very similar but slightly modified wording of the same query might succeed.

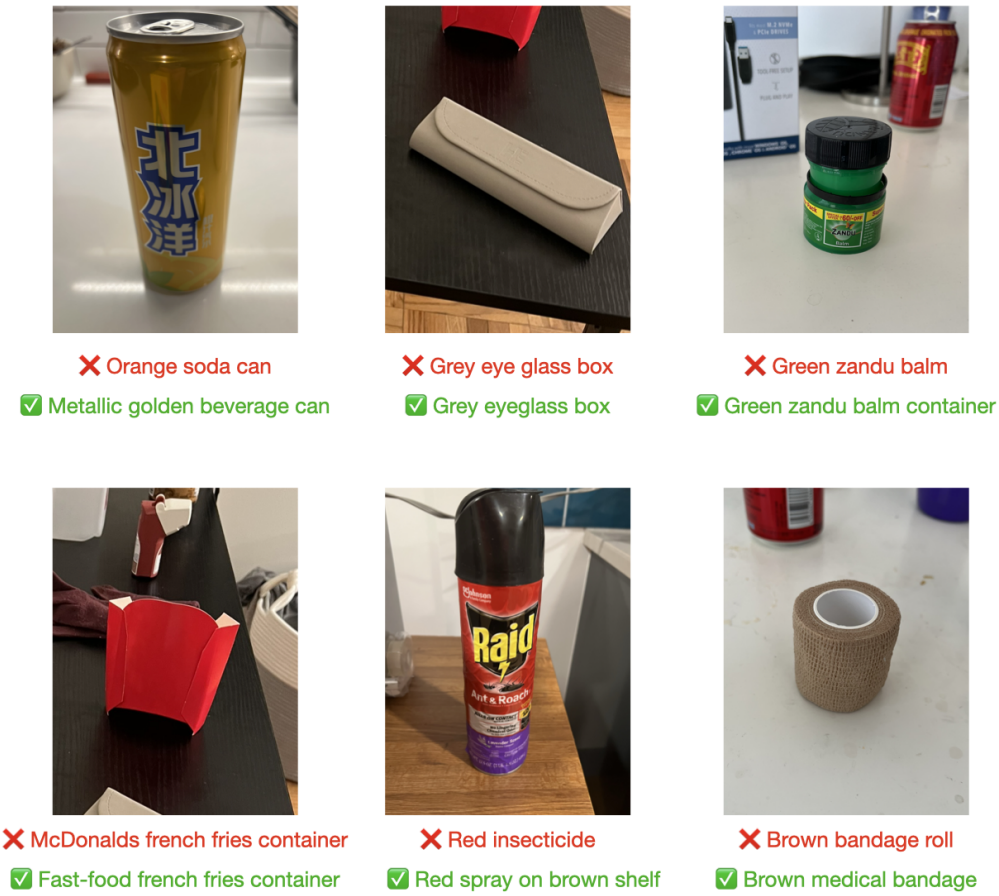


Figure 10.7: Samples of failed or ambiguous language queries into our semantic memory module. Since the memory module depends on pretrained large vision language model, its performance shows susceptibility to particular "incantations" similar to current LLMs.

Generally, this has been the case for scenes where there are multiple visually or semantically similar objects, as shown in the figure. There are other cases where some queries may pass while other very similar queries may fail. An interactive system that gets confirmation from the user as it retrieves an object from memory would avoid such issues.

10.3.4.2 Grasping module limitations

One failure mode of our manipulation module comes from executing grasps from a pre-trained manipulation model's output based on a single RGB-D image. Moreover, this model wasn't even designed for the Hello Robot: Stretch gripper. As a result, sometimes the proposed grasps are

unreliable or unrealistic (Figure 10.8).

Sometimes, the grasp is infeasible given the robot joint limits, or is simply too far from the robot body. Developing better grasp perception models or heuristics will let us sample better grasps for a given object.

In other cases, the model generates a good grasp pose, but as the robot is executing the grasping primitive, it collides with some minor environment obstacle. Since we apply the same grasp trajectory in every case instead of planning the grasp trajectory, some such failures are inevitable. Grasping models that generates a grasp trajectory as well as a pose may solve such issues.

Finally, our grasping module categorically struggles with flat objects, like chocolate bars and books, since it's difficult to grasp them off a surface with a two-fingered gripper.

10.3.4.3 Robot hardware limitations

While our robot of choice, a Hello Robot: Stretch, is able to pick-and-drop a variety of objects, certain hardware limitations also dictate what our system can and cannot manipulate. For example, the fully extended robot arm has a 1 kg payload limit, and thus our method is unable to pick objects like a full dish soap bottle. Similarly, objects that are far from navigable floor space, i.e. in the middle of a bed, or on high places, are difficult for the robot to reach because of the reach limits of the arm. The robot hardware or the RealSense camera can occasionally get miscalibrated over time, especially during continuous home operations. This miscalibration can lead to manipulation errors since that module requires hand-eye coordination in the robot. The robot base wheels have small diameters and in some cases struggle to move smoothly between carpet and floor.

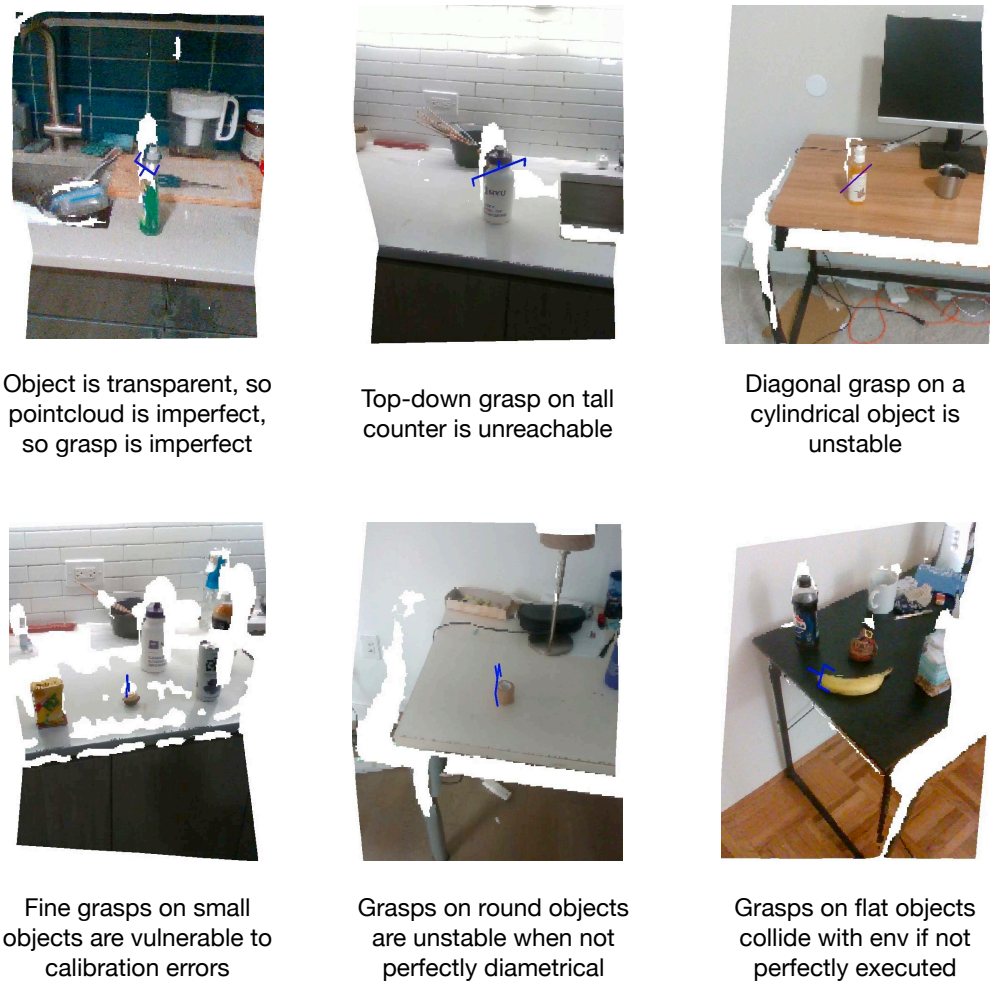


Figure 10.8: Samples of failures of our manipulation module. Most failures stem from using only a single RGB-D view to generate the grasp and the limiting form-factor of a large two-fingered parallel jaw gripper.

10.4 Related Works

10.4.1 Vision-Language models for robotic navigation

Early applications of pre-trained open-knowledge models in robotics has been in open-vocabulary navigation. Navigating to various objects is an important task which has been looked at in a wide range of previous works [Mousavian et al. 2019b; Yenamandra et al. 2023b; Chang et al. 2023], as well as in the context of longer pick-and-place tasks [Blukis et al. 2022; Min et al. 2021]. However,

these methods have generally been applied to relatively small numbers of objects [Deitke et al. 2022]. Recently, Objaverse [Deitke et al. 2023] has shown navigation to thousands of object types, for example, but much of this work has been restricted to simulated or highly controlled environments.

The early work addressing this problem builds upon representations derived from pre-trained vision language models, such as SemAbs [Ha and Song 2022], CLIP-Fields [Shafiullah et al. 2023a], VLMaps [Huang et al. 2023b], NLMap-SayCan [Chen et al. 2022a], and later, ConceptFusion [Jatavallabhula et al. 2023] and LERF [Kerr et al. 2023]. Most of these models show object localization in pre-mapped scenes, while CLIP-Fields, VLMaps, and NLMap-SayCan show integration with real robots for indoor navigation tasks. USA-Nets [Bolte et al. 2023] extends this task to include an affordance model, navigating with open-vocabulary queries while doing object avoidance. ViNT [Shah et al. 2023] proposes a foundation model for robotic navigation which can be applied to vision-language navigation problems. More recently, GOAT [Chang et al. 2023] was proposed as a modular system for "going to anything" and navigating to any object in any environment given either language or image queries. ConceptGraphs [Gu et al. 2023] proposed an open scene graph representation capable of handling complex queries using LLMs. Any such open-vocabulary embodied model has the potential to improve modular systems like `OK-Robot`.

10.4.2 Pretrained robot manipulation models

While humans can frequently look at objects and immediately know how to grasp it, such grasping knowledge is not easily accessible to robots. Over the years, there has been many works that has focused on creating such a general robot grasp generation model [Pinto and Gupta 2016; Gupta et al. 2018; Mahler et al. 2017a, 2018; Kalashnikov et al. 2018; Qin et al. 2019; Mousavian et al. 2019a] for arbitrary objects and potentially cluttered scenes via learning methods. Our work focuses on more recent iterations of such methods [Sundermeyer et al. 2021; Fang et al. 2023c] that

are trained on large grasping datasets [Eppner et al. 2021; Fang et al. 2020]. While these models only perform one task, namely grasping, they predict grasps across a large object surface and thus enable downstream complex, long-horizon manipulation tasks [Goyal et al. 2022b; Singh et al. 2023; Liu et al. 2022].

More recently, there is a set of general-purpose manipulation models moving beyond just grasping [Shridhar et al. 2023; Parashar et al. 2023; Shafiullah et al. 2022; Cui et al. 2022; Gervet et al. 2023b]. Some of these works perform general language-conditioned manipulation tasks, but are largely limited to a small set of scenes and objects. HACMan [Zhou et al. 2023b] demonstrates a larger range of object manipulation capabilities, focused on pushing and prodding. In the future, such models could expand the reach of our system.

10.4.3 Open vocabulary robot systems

Many recent works have worked on language-enabled tasks for complex robot systems. Some examples include language conditioned policy learning [Shridhar et al. 2022, 2023; Lynch et al. 2020; Lynch and Sermanet 2021], learning goal-conditioned value functions [Brohan et al. 2023b; Huang et al. 2023c], and using large language models to generate code [Liang et al. 2023; Wang et al. 2023a; Singh et al. 2023]. However, a fundamental difference remains between systems which aim to operate on arbitrary objects in an open-vocab manner, and systems where one can specify one among a limited number of goals or options using language. Consequently, Open-Vocabulary Mobile Manipulation has been proposed as a key challenge for robotic manipulation [Yenamandra et al. 2023b]. There has previously been efforts to build such a system [Yokoyama et al. 2023; Stone et al. 2023]. However, unlike such previous work, we try to build everything on an open platform and ensure our method can work without having to re-train anything for a novel home. Recently, UniTeam [Melnik et al. 2023] won the 2023 HomeRobot OVMM Challenge [Yenamandra et al. 2023a] with a modular system doing pick-and-place to arbitrary objects, with a zero-shot

generalization requirement similar to ours.

In parallel, recently, there have been a number of papers doing open-vocabulary manipulation using GPT or especially GPT4 [OpenAI 2023]. GPT4V can be included in robot task planning frameworks and used to execute long-horizon robot tasks, including ones from human demonstrations [Wake et al. 2023]. ConceptGraphs [Gu et al. 2023] is a good recent example, showing complex object search, planning, and pick-and-place capabilities to open-vocabulary objects. SayPlan [Rana et al. 2023] also shows how these can use used together with a scene graph to handle very large, complex environments, and multi-step tasks; this work is complementary to ours, as it doesn't handle how to implement pick and place.

10.5 Limitations, Open Problems and Requests for Research

While our method shows significant success in completely novel home environments, it also shows many places where such methods can improve. In this section, we discuss a few of such potential improvement in the future.

10.5.1 Live semantic memory and obstacle maps

All the current semantic memory modules and obstacle map builders build a static representation of the world, without a good way of keeping it up-to-date as the world changes. However, homes are dynamic environments, with many small changes over the day every day. Future research that can build a dynamic semantic memory and obstacle map would unlock potential for continuous application of such pick-and-drop methods in a novel home out of the box.

10.5.2 Grasp plans instead of proposals

Currently, the grasping module proposes generic grasps without taking the robot's body and dynamics into account. Similarly, given a grasp pose, often the open loop grasping trajectory collides with environmental obstacles, which can be easily improved by using a module to generate grasp plans rather than grasp poses only.

10.5.3 Improving interactivity between robot and user

One of the major causes of failure in our method is in navigation: where the semantic query is ambiguous and the intended object is not retrieved from the semantic memory. In such ambiguous cases, interaction with the user would go a long way to disambiguate the query and help the robot succeed more often.

10.5.4 Detecting and recovering from failure

Currently, we observe a multiplicative error accumulation between our modules: if any of our independent components fail, the entire process fails. As a result, even if our modules each perform independently at or above 80% success rate, our final success rate can still be below 60%. However, with better error detection and retrying algorithms, we can recover from much more single-stage errors, and similarly improve our overall success rate [Melnik et al. 2023].

10.5.5 Robustifying robot hardware

While Hello Robot - Stretch [Kemp et al. 2022] is an affordable and portable platform on which we can implement such an open-home system for arbitrary homes, we also acknowledge that with robust hardware such methods may have vastly enhanced capacity. Such robust hardware

may enable us to reach high and low places, and pick up heavier objects. Finally, improved robot odometry will enable us to execute much more finer grasps than is possible today.

Postscript

`OK-Robot` as a method can be read as a response to contemporary end-to-end approaches to mobile manipulation that were necessarily limited in the environments where they could operate. Showing that a modular design can readily deploy to many diverse environments without having to train or fine-tune a single parameter was a boast. At the same time understanding the ways in which such elaborate, modular systems fail due to the the many corner cases and incompatibility between modules was a necessary step to establish the science of mobile manipulation. In the long run, we as the field need to run a lot more longitudinal study of a similar nature if we are to bring general robots into our everyday environments to understand all the different, long-tail ways our robots may fail as soon as the rubber hits the road.

Acknowledgments

This work was co-led by Peiqi Liu and Yaswanth Orru, co-authored with Jay Vakil and Chris Paxton, and co-advised with Lerrel Pinto. NYU authors are supported by grants from Amazon, Honda, and ONR award numbers N00014-21-1-2404 and N00014-21-1-2758. NMS is supported by the Apple Scholar in AI/ML Fellowship. LP is supported by the Packard Fellowship. Our utmost gratitude goes to our friends and colleagues who helped us by hosting our experiments in their homes. Finally, we thank Siddhant Haldar, Paula Pascual and Ulyana Piterbarg for valuable feedback and conversations.

11 | Online Dynamic Spatio-Semantic Memory for Open World Mobile Manipulation: DynaMem

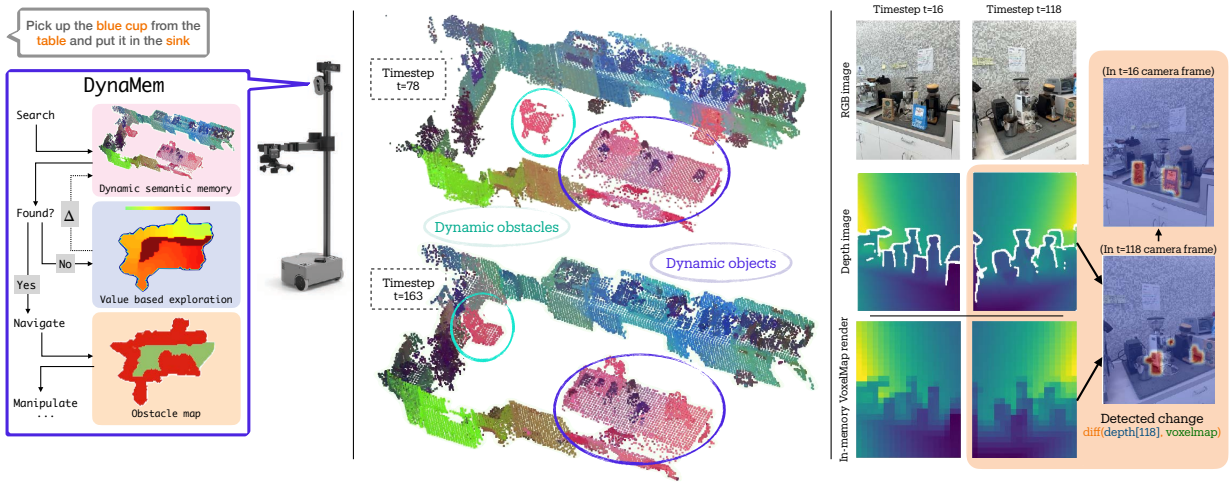


Figure 11.1: An illustration of how DynaMem, our online dynamic spatio-semantic memory responds to open vocabulary queries in a dynamic environment. During operation and exploration, DynaMem keeps updating its semantic map in memory. DynaMem maintains a voxelized pointcloud representation of the environment, and updates with dynamic changes in the environment by adding and removing points.

11.1 Introduction

Recent advances in robotics have made it possible to deploy robots in real world settings to tackle the open vocabulary mobile manipulation (OVMM) problem [Yenamandra et al. 2023b]. Here, the robots are tasked with navigating in unknown environments and interacting with objects following open vocabulary language instructions, such as "Pick up **X** from **Y** and put it in **Z**", where X, Y, and Z could be any object name or location. The two most common approaches to tackling OVMM are using policies trained in simulation and deploying them in the real world [Ehsani et al. 2023; Ramrakhya et al. 2023; Zeng et al. 2024], or training modular systems that combine open vocabulary navigation (OVN) [Shafiullah et al. 2023a; Gu et al. 2024; Maggio et al. 2024; Kerr et al. 2023] with different robot manipulation skills [Liu et al. 2024c; Qiu et al. 2024; Bolte et al. 2023; Chang et al. 2023; Werby et al. 2024]. Modular systems enjoy greater efficiency and success in real-world deployment [Gervet et al. 2023a] as they can directly leverage advances in vision and language models [Liu et al. 2024c; Chang et al. 2023], and are able to handle more diverse and out-of-domain environments with no additional training.

However, as recent analysis has shown, the primary challenge in deploying modular OVMM is that limitations of a module propagate to the entire system [Liu et al. 2024c]. One key module in any OVMM system is the open vocabulary navigation (OVN) module responsible for navigating to goals in the environment. While many such OVN systems have been proposed in the literature [Yenamandra et al. 2023b; Kerr et al. 2023; Shafiullah et al. 2023a; Bolte et al. 2023; Qiu et al. 2024; Liu et al. 2024c; Gu et al. 2024; Maggio et al. 2024; Chang et al. 2023; Werby et al. 2024], they share a common limitation: they assume static, unchanging environments. Contrast this with the real world, where environments change and objects are moved by either robots or humans. Making such a restrictive assumption thus limits these systems' applicability in real-world settings. The primary reason behind this assumption is the lack of an effective dynamic spatio-semantic

memory that can adapt to both addition and removal of objects and obstacles in the environment online.

In this work, we propose a novel spatio-semantic memory architecture, Dynamic 3D Voxel Memory (DynaMem), that can adapt online to changes in the environment. DynaMem maintains a voxelized pointcloud representation of an environment and adds or removes points as it observes the environment change. Additionally, it supports two different ways to query the memory with natural language: a vision-language model (VLM) featurized pointcloud, and a multimodal-LLM (mLLM) QA system. Finally, DynaMem enables efficient exploration in changing environments by offering a dynamic obstacle map and a value-based exploration map that can guide the robot to explore unseen, outdated, or query-relevant parts of the world.

We evaluate DynaMem as a part of full open-vocabulary mobile manipulation stack in three real world environments with multiple rounds of changes and manipulating multiple non-stationary objects, improving the static baseline by more than 2× (70% vs. 30%). Additionally, we identify an obstacle in efficiently developing dynamic spatio-semantic memory, namely the lack of dynamic benchmarks, since many OVN systems use static simulated environments [Chen et al. 2020b; Dai et al. 2017] or static datasets [Yadav et al. 2023; Baruch et al. 2021]. We address this by developing a new dynamic benchmark, DynaBench. It consists of 9 different environments, each changing over time. We ablate our design choices in this benchmark. To the best of our knowledge, DynaMem is the first spatio-semantic memory structure supporting both adding and removing objects.

11.2 Related Works

11.2.1 Open Vocabulary Mobile Manipulation (OVMM)

Navigating to arbitrary goals in open ended environments and manipulating them has become a key challenge in robotic manipulation [Yenamandra et al. 2023a; Yokoyama et al. 2023]. This line of query follows Open-Vocabulary Navigation systems [Shafiullah et al. 2023a; Huang et al. 2023b], which builds upon prior object and point goal navigation literature [Gervet et al. 2023a; Majumdar et al. 2020; Krantz et al. 2022; Hahn et al. 2021; Chaplot et al. 2020; Yokoyama et al. 2021; Zhao et al. 2021; Batra et al. 2020; Chang et al. 2023] which attempted navigation to points, or fixed set of objects and object categories. OVMM is a naturally harder challenge as it requires an ability to handle arbitrary queries, and "navigation to manipulation" transfer – which means unlike pure navigation, the robot needs to get close to the environment objective and obstacles. In the OVMM challenge [Yenamandra et al. 2023a], modular solutions such as [Yenamandra et al. 2023b; Melnik et al. 2023; Werby et al. 2024] outperformed the competition. More recently, OK-Robot [Liu et al. 2024c] performed extensive real-world evaluations of the challenges in OVMM and demonstrated a system that achieves 58.5% success rate in static home environments. We extend this work by enabling manipulation in changing environments.

11.2.2 Spatio-semantic Memory

Early works in spatio-semantic memory [Henry et al. 2012; Bowman et al. 2017; Zhang et al. 2018a; Ma et al. 2017; Chaplot et al. 2020] created semantic maps for limited categories based on mostly ad-hoc deep neural networks. Later work builds upon representations derived from pre-trained vision language models, such as [Ha and Song 2022; Shafiullah et al. 2023a; Huang et al. 2023b; Chen et al. 2022a; Jatavallabhula et al. 2023; Kerr et al. 2023; Gu et al. 2024; Maggio

et al. 2024]. These works use a voxel map or neural feature field as their map representation. Some recent models [Ji et al. 2024; Shorinwa et al. 2024] have used Gaussian splats [Kerbl et al. 2023] to represent semantic memory for manipulation. Most of these models show object localization in pre-mapped scenes, while CLIP-Fields [Shafiullah et al. 2023a], huang2023visual [Huang et al. 2023b], and NLMap-SayCan [Chen et al. 2022a] show integration with real robots for indoor navigation tasks. Some recent works [Bolte et al. 2023; Wang et al. 2023b; Qiu et al. 2024] extend this task to include an affordance model or manipulation primitives. Our work builds upon the voxel map based spatio-semantic memory literature and extends them to dynamic environments where both objects and obstacles can change over time. Concurrent to our work, DovSG [Yan et al. 2024b] looks at dynamic semantic scene graphs. As scene graphs deal with an object level abstraction, DovSG needs to handle object merging, association, and deduplication explicitly, which are all handled implicitly in DynaMem.

11.2.3 Mapping and Navigating Dynamic Environments

For robot navigation, Simultaneous Localization and Mapping (SLAM) [Durrant-Whyte and Bailey 2006] methods are crucial. However, practical SLAM instances based on voxels [Song et al. 2024; Shi et al. 2021], objects [McCormac et al. 2018; Krishna et al. 2023], landmark [Bowman et al. 2017; Michael et al. 2022], NeRF [Maggio et al. 2023; Rosinol et al. 2023], and Gaussian splats [Matsuki et al. 2024; Yan et al. 2024a] tend to make the simplifying assumption that the world is static. Some sparse SLAM methods improve on dynamic environments by estimating underlying state [Qiu et al. 2022; Cui and Ma 2019; Brasch et al. 2018; Yu et al. 2018; Song et al. 2022; Yu et al. 2021; Bescos et al. 2018; Schmid et al. 2024; Virgolino Soares et al. 2023] or explicitly modeling moving objects [Bescos et al. 2021; Henein et al. 2020; Henning et al. 2022]. Some methods also forego a map and rely on reactive policies to navigate dynamic environments [Haviland et al. 2022; Brohan et al. 2023b; Du et al. 2022; Wong et al. 2022; Uppal et al. 2024], although they generally tackle

local movement and not global navigation. Our work relies on SLAM systems that are stable under environment dynamics, and focuses on building a dynamic semantic memory based off of online exploration and observations.

11.3 Method

In this section, we define our problem setup, and then describe our online, dynamic spatio-semantic memory for open world, open vocabulary mobile manipulation.

11.3.1 Problem Statement

We create our algorithm, DynaMem, to solve open vocabulary mobile manipulation (OVMM) problems in an open, constantly changing world. The goal in OVMM is for a mobile robot to execute a series of manipulation commands given arbitrary language goals. We assume the following requirements for the memory module for dynamic, online operation:

- **Observations:** The mobile robot is equipped with an on-board RGB-D camera, and unlike prior work [Liu et al. 2024c], doesn't start with a map of the environment. Rather, the robot explores the world and use the online observed sequence of posed RGB-D images to build its map.
- **Environment dynamism:** The environment can change without the knowledge of the robot.
- **Localization queries:** Given a natural language query (i.e. "teddy bear"), the memory module has to return the 3D location of the object or determine that the object doesn't exist in the scene observed thus far.
- **Obstacle queries:** The memory module must determine whether a point in space is occupied by an obstacle. Both the location of the objects and obstacles can move, previous observations often contradict each other and must be resolved by the memory.

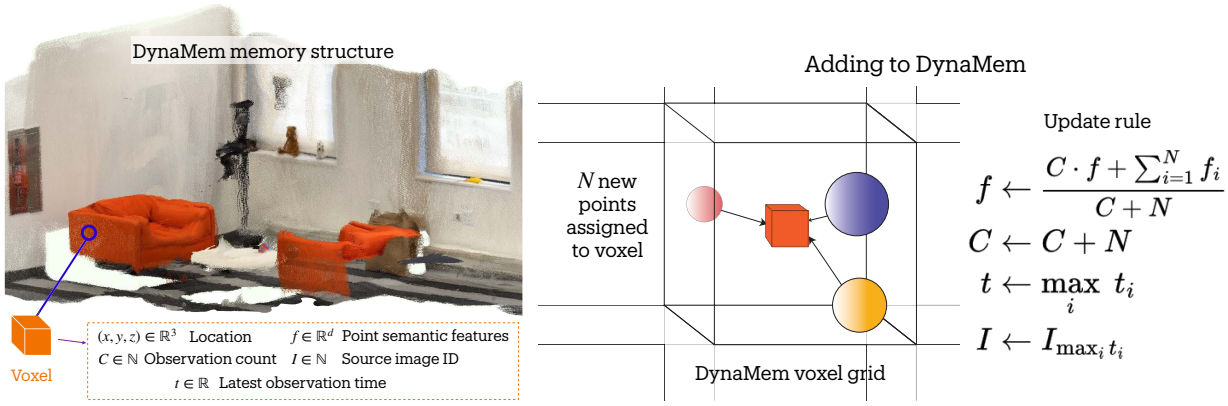


Figure 11.2: (Left) DynaMem keeps its memory stored in a sparse voxel grid with associated information at each voxel. (Right) Updating DynaMem by adding new points to it, alongside the rules used to update the stored information.

Note the significant upgrade in challenge in multiple facets compared to prior work [Yenamandra et al. 2023b; Kerr et al. 2023; Shafiullah et al. 2023a; Bolte et al. 2023; Qiu et al. 2024; Liu et al. 2024c; Gu et al. 2024; Maggio et al. 2024; Chang et al. 2023; Werby et al. 2024]: almost no prior work dealing with open-vocabulary queries support dynamic environments with both addition and deletion, some assumes access to prior map data, and many don't handle *negative* results, i.e. objects not found in memory, and instead return the best match.

11.3.2 Dynamic 3D Voxel Map

Our answer to the challenge posed in the Section 11.3.1 is DynaMem. DynaMem is an evolving sparse voxel map with associated information stored at each voxel, as shown in Figure 11.2. In each non-empty voxel, alongside its 3D location (x, y, z), we also store the observation count C (how many times that voxel was observed), source image ID I (which image the voxel was backprojected from), a high-dimensional semantic feature vector f coming from a VLM like CLIP [Radford et al. 2021] or SigLIP [Zhai et al. 2023], and the latest observation time, t, in seconds.

To make this data structure dynamic, we describe the process with which we add and update with new observations and remove outdated objects and associated voxels.

11.3.2.1 Adding Points

When the robot receives a new set of observations, i.e. RGB-D images with global poses, we convert them to 3D coordinates in a global reference frame, and generate a semantic feature vector for each point. The global coordinates are calculated from the global camera pose and the backprojected depth image using the known camera transformation matrix. We calculate the point-wise image feature by first converting the images to object patches by using a segmentation model such as SAM-v2 [Ravi et al. 2024], and then aggregating each patch feature over the output of a vision-language models like CLIP [Radford et al. 2021] or SigLIP [Zhai et al. 2023]. For more details about image-to-feature vector mapping, we refer to earlier works [Shafiullah et al. 2023a; Liu et al. 2024c; Kerr et al. 2023]. Once we have calculated the points and associated features, we cluster the new points and assign them to the nearest voxel grids. In Figure 11.2, we show how each voxel's metadata is updated. The count keeps track of the total number of assigned points to each voxel grid, and the feature vector keeps track of the weighted average of all feature vectors assigned to that voxel. Finally, the observation time and image ID are updated to keep track of the latest observation contributing to a particular voxel. If a voxel was empty before assignment, we assume its count $C = 0$ and feature vector $f = \overrightarrow{0}$.

11.3.2.2 Removing Points

When an object is moved or removed, its associated voxels in DynaMem may get removed. We use ray-casting to find the outdated voxels. The operation follows a simple principle: if a voxel falls within the frustum between the camera plane and the associated view point cloud, that voxel must be unoccupied. To reduce the impact of the depth noise at long range, we don't consider any pixel whose associated depth value is over 2m. We illustrate a simplified 2D representation of this algorithm in Figure 11.3. In practice, to speed up the intersection between the sparse voxelmap and the view frustum, we project each existing voxel to the camera plane and calculate the camera

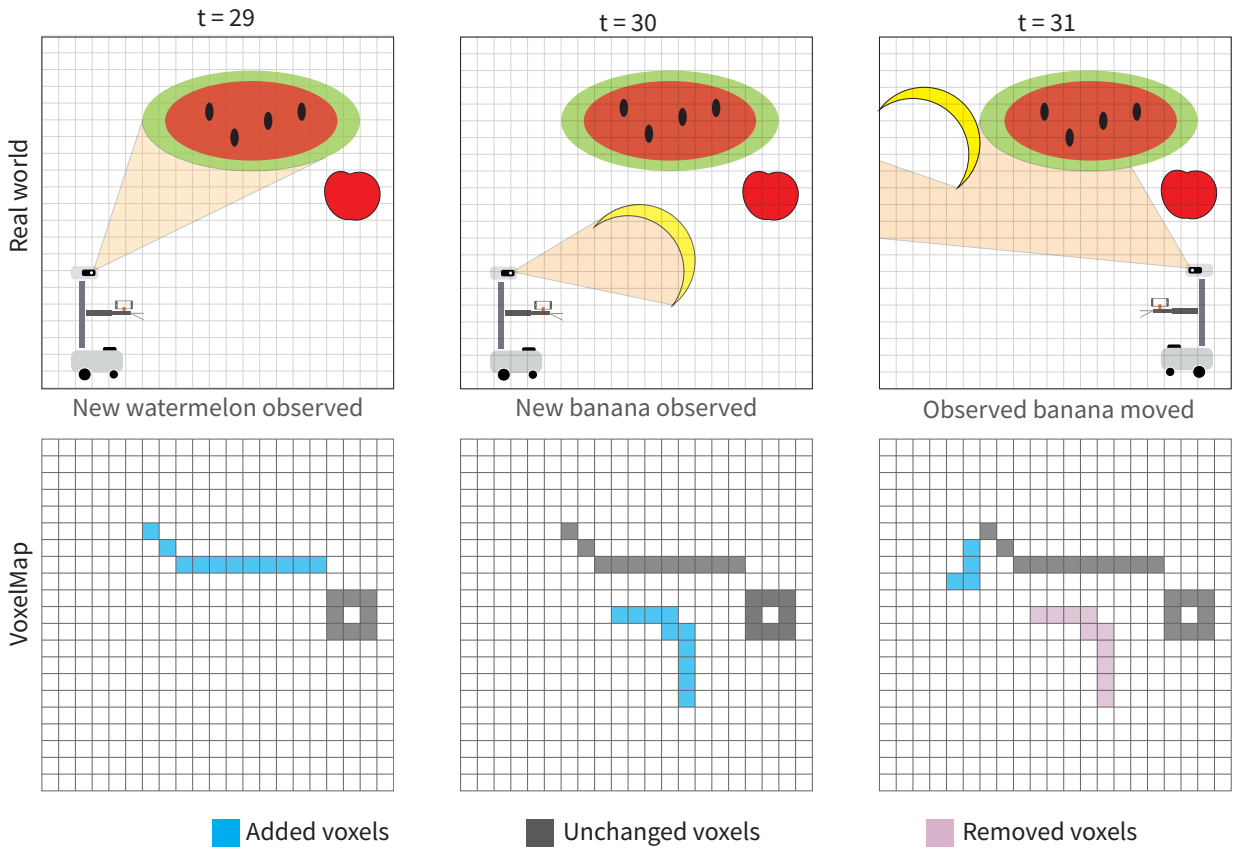


Figure 11.3: A high-level, 2D depiction of how adding and removing voxels from the voxel map works. New voxels are included which are in the RGB-D cameras view frustum, and old voxels that should block the view frustum but does not are removed from the map.

distance. If the image height and width are (H, W), the depth image is $\mathbf{D}$, and a certain voxel is projected to points (h, w) in the camera plane with depth d, it gets removed if both Eq. 11.1 and 11.2 hold.

$$(h, w) \in [0, H] \times [0, W] \tag{11.1}$$

$$d \in \left(0, \min(2, \mathbf{D}\left[h, w\right] + \epsilon)\right) \tag{11.2}$$

Where Eq. 11.1 ensures that the point falls within the camera view, and Eq 11.2 ensures that (a) the depth $d > 0$, or the object is in front of camera, (b) $d < 2$m, or the voxel isn't too far away from the camera, and (c) $d < \mathbf{D}[h, w]$ denoting the voxel is between the camera and the currently visible object.

11.3.3 Querying DynaMem for Object Localization

As described in Section 11.3.1, we define the object localization or 3D visual grounding problem as a function mapping a text query and posed RGBD images to either the 3D coordinate of the query object, or $\emptyset$ if the object is not in the scene. Unlike previous work, we abstain from returning a location when an object is not found. To enable this, we factor this grounding problem into two sub-problems. The first is finding the latest image where the queried object could have appeared. The second is identifying whether the object is actually present in that image. For the first sub-problem, we introduce two alternate approaches of visual grounding: one using the intrinsic semantic features of DynaMem, and another using state-of-the-art multimodal LLMs such as GPT-4o [Team 2024] and Gemini 1.5 Pro [Google 2024].

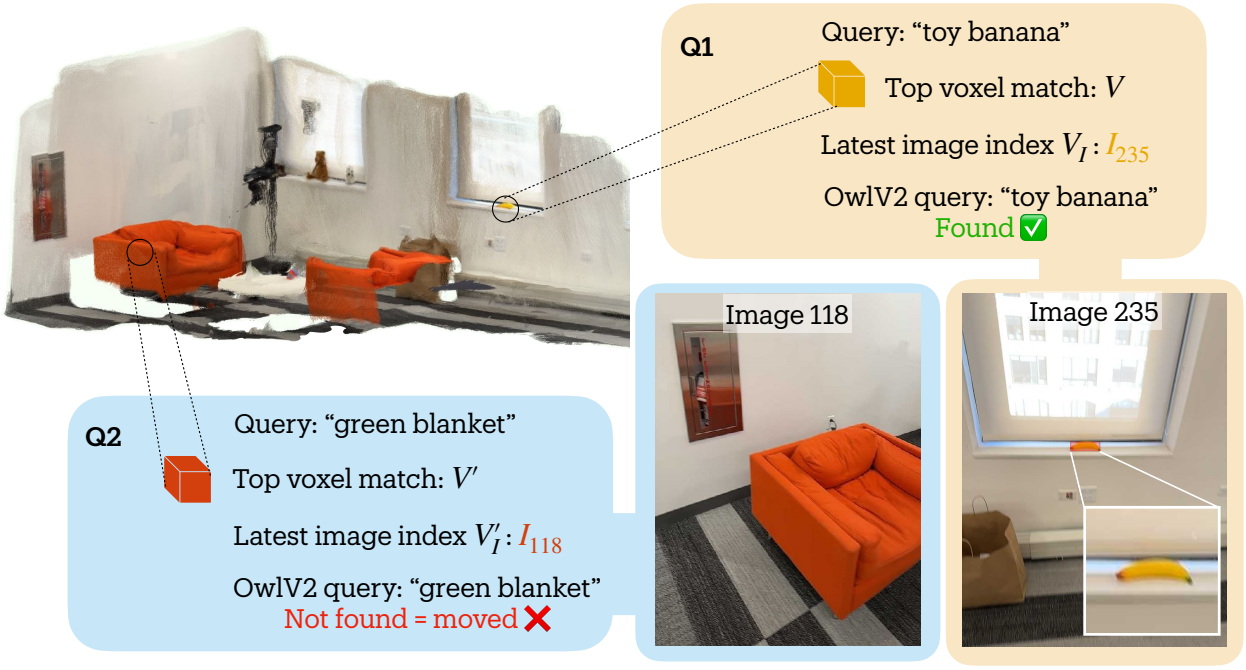


Figure 11.4: Querying DynaMem with a natural language query. First, we find the voxel with the highest alignment to the query. Next, we find the latest image of that voxel, and query with an open-vocabulary object detector to confirm the object location or abstain.

11.3.3.1 Embedded Vision Language Features

Vision Language Models (VLMs) such as CLIP [Radford et al. 2021] and SigLIP [Zhai et al. 2023] possess an ability to embed both images and languages into the same latent space, where the similarity between an image and a text object can be calculated by simply taking the dot product between the two latent representation vectors. We use this property of the embedding vectors to query our voxel map with open-vocabulary text queries. As described in Section 11.3.2, we convert the incoming images to point-wise image features, and embed them into our voxels. When we have a new language query, we calculate its latent embedding using the VLM text encoder, and find the voxel whose feature has the highest dot product with the text embedding. Once we find the right voxel, we simply retrieve its associated latest image from our data structure as shown in Figure 11.4.

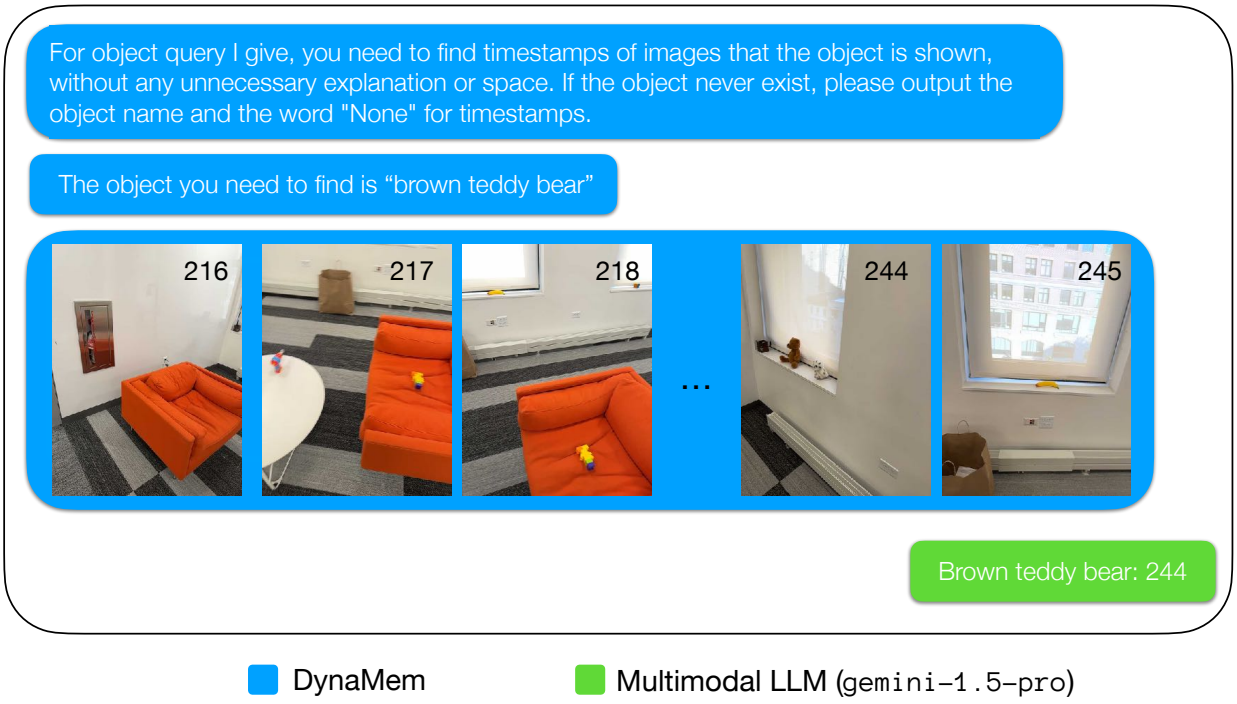


Figure 11.5: The prompting system for querying multimodal LLMs such as GPT-4o or Gemini-1.5 for the image index for an object query.

As a bonus feature, our VoxelMap can also return $k > 1$ possible objects and associated images for a single query. We do this by using a DBSCAN clustering of voxels similar to [Yang et al. 2023], and returning the images associated with the most aligned voxel in top-k clusters.

11.3.3.2 Multimodal Large Language Models (mLLMs)

For this approach, we note that the problem of finding the latest image where an object may appear is similar to the problem of visual question-answer (VQA) [Antol et al. 2015]. Since we fully rely on pretrained models to build our map, we pose this multi-image VQA problem as an mLLM QA problem similar to OpenEQA [Majumdar et al. 2024].

We show in Figure 11.5 how we query the mLLMs to solve the visual grounding query. We give the model a sequence of our latest environment observations images and ask the model for the index of the last image where the queried object was observed. We additionally instruct

the model to respond "None" if the object was not observed in any image. Note that, unlike OpenEQA [Majumdar et al. 2024], we only pass the RGB images to the mLLM, and not the depth or camera pose. Similarly, we only ask for an image index, and not a full textual answer.

One important hyperparameter for this mLLM query method is the maximum number of images included in the prompt. Longer context needs longer processing time and potentially includes outdated information, while short context might not include all information and thus will miss objects. We optimize the context by excluding completely outdated images: all images I with no voxel pointing to them are deleted. This filtering increases mLLM context utilization. We set Gemini as our base model and 60 as our query image limit since Gemini context can fit 60 images, which is twice as large as GPT-4o's context size.

11.3.3.3 Combining the Approaches

From the discussion above, and from our real-world experiments as shown in Section 11.4, we see that the downsides of these two methods in practice are somewhat complementary. The VoxelMap can easily process a large number of observation images over time, but it struggles to disambiguate between multiple similar but not quite same objects. On the other hand, mLLM based methods can distinguish between fine differences in query objects, but can only handle few images at a time. As a result, we come up with a *hybrid* approach – taking advantage of the best of both methods.

For this approach, we build and process the VoxelMap as usual. For the hybrid querying, parametrized by an integer k, we retrieve top k candidate images from the VoxelMap for the given query. Then, we pass those k images into the mLLM, and query them as usual for the mLLM approach. The mLLM answers with the latest image where the query object may appear, which we then use for the downstream processing. Note that, this approach generalizes both of our individual approaches: when $k = 1$, it converges to the VLM-feature-only approach, and when $k \to \infty$ it converges to the mLLM-only approach.

11.3.3.4 Handling Absence of Queried Object

Several previous methods [Shafiullah et al. 2023a; Kerr et al. 2023; Liu et al. 2024c] assume that the queried object is always present in the scene, and always responds with the object that is the best match to the query. However, this often results in high false-positive failure cases. For example, in a scene with no red cups and a blue cup, the method may respond with the location of the blue cup in response to the query "red cup".

For this reason, we locate objects in two stages. First, we find the best candidate image where the object may have been seen (Section 11.3.3.1). Then, we use an open-vocabulary object detector model such as OWL-v2 [Minderer et al. 2024] to search that image for the queried object (Figure 11.4). If we don't find the queried object, we assume that the object has either moved, or the response from the voxelmap or mLLM was inaccurate, and respond with "object not found". If the open-vocabulary object detector returns an object bounding box, we find the median pixel from the object mask and return its 3D location.

11.3.4 Robot Navigation and Exploration

To navigate in a real-world environment, robots use an obstacle map in conjunction with a navigation algorithm like A* in [Huang et al. 2023b; Liu et al. 2024c]. We use a simple voxel-projection strategy to build an obstacle map. Due to the depth observation noise, we simply set a threshold for the ground (0.2m for our experiments), and project all the voxels above that z-threshold as the obstacles in our map. The voxels below the threshold are projected into the 2D obstacle map as navigable points. Finally, the points in the map that are not marked as either obstacle or navigable are marked as explorable points.

11.3.4.1 Exploration Primitives

Since our robot does not start with an environment map, it explores the environment with frontier based methods to build the map. We can further accelerate this process by providing exploration guidance. Based on the current status of the map, DynaMem provides an exploration value function to accelerate the exploration process both for building and updating the map.

We provide two value-based exploration maps: one time-based, and one semantic-similarity-based [Yokoyama et al. 2024]. The time-based value map prioritizes the least-recently seen points. If the current time is T, and the last-seen time of voxel (x, y, z) is $t_{x,y,z}$, the temporal value map $\mathbb{V}_T$ is expressed as:

$$\mathbf{T}^*[x, y] = \max_z (T - t_{x,y,z})$$
$$\mathbb{V}_T[x, y] = \sigma\left(-\beta_T\left(\mathbf{T}^*[x, y] - \mu_T\right)\right)$$

where β_T, μ_T are hyper-parameters and σ is the sigmoid function. Similarly, if the VLM feature at voxel (x, y, z) is $f_{x,y,z}$, and the VLM feature for the language query is f_q, then the similarity-based value map $\mathbb{V}_S$ is be expressed as:

$$\mathbf{S}^*[x, y] = \max_z (f_q \cdot f_{x,y,z})$$
$$\mathbb{V}_S[x, y] = \sigma\left(-\beta_S\left(\mathbf{S}^*[x, y] - \mu_S\right)\right)$$

where once again β_S, μ_S are hyperparameters. We may also linearly combine $\mathbb{V}_T, \mathbb{V}_S$ to balance our exploration between last seen time and semantic similarity.

Finally, since the environment may be dynamic, we convert our navigation algorithm from open-loop to closed-loop. The robot, instead of executing the entire navigation plan generated by A*, stops after the first seven waypoints (approx. 0.7 to 1 meters). Then, the robot scans the

environment, updates the map, and moves according to a new plan. The robot repeats these steps until its distance to the target is lower than a predefined threshold.

11.4 Experiments

We evaluate our method, DynaMem, on a Hello Robot: Stretch SE3 in real world environments, and perform a series of ablation experiments in an offline benchmark.

11.4.1 Real-world Experiments

We evaluate DynaMem and its impact on open-vocabulary mobile manipulation in three real-world dynamic environments (Figure 11.6). In each environment, we set up multiple objects as potential manipulation targets, change the environment in three rounds, and execute 10 pick-and-drop queries over the rounds We use the Hello Stretch SE3 as our mobile robot platform, and use its head-mounted Intel RealSense D435 RGB-D camera to collect the input data.

To build a complete pick-and-drop system around DynaMem, we follow the system architecture in OK-Robot [Liu et al. 2024c]. In particular, we use the AnyGrasp [Fang et al. 2023c] based open-vocabulary grasp system and use the heuristic based dropping system. However, we use DynaMem's exploration primitives let the robot build the map of the environment and allow the robot to explore when an object is not found in the memory.

As a baseline, we compare with OK-Robot [Liu et al. 2024c], a state-of-the-art method for OVMM. OK-Robot uses a static voxelmap as its memory representation, and thus it highlights the importance of dynamic memory for OVMM in a changing environment. For DynaMem, we run two variations of the algorithm in the real world: one with VLM-feature based queries and one with mLLM-QA based queries.

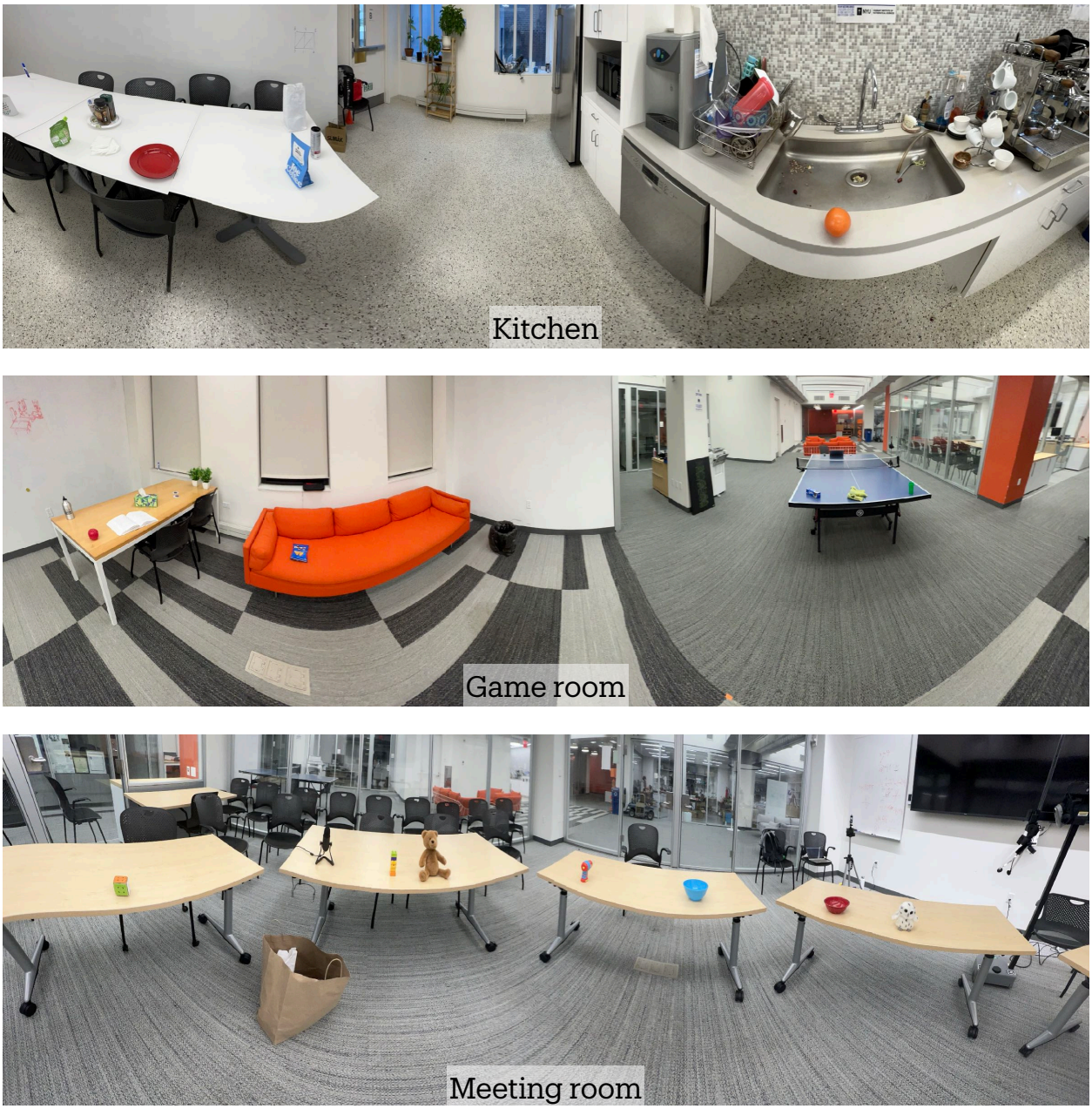
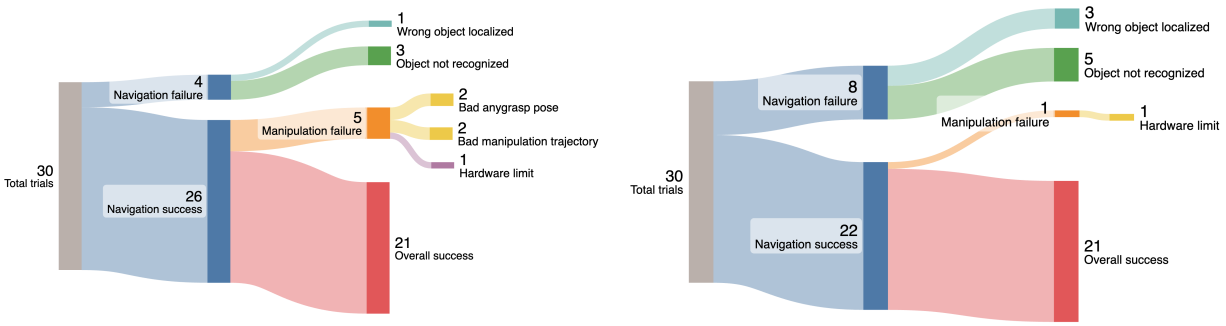


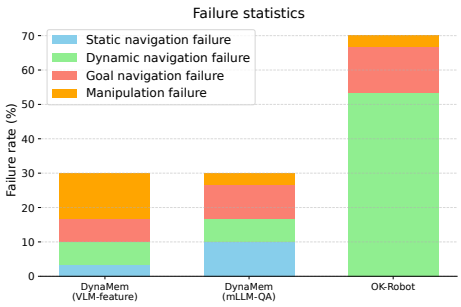
Figure 11.6: Real robot experiments in three different environments: kitchen, game room, and meeting room. In each environment, we modify the environment thrice and run 10 pick-and-drop queries.

11.4.1.1 Results

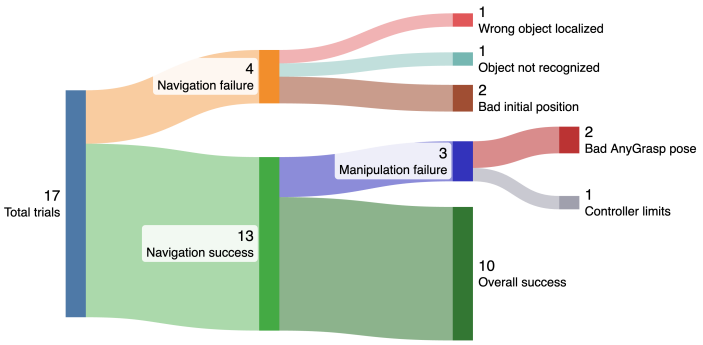
Our experiments in three dynamic environments and with 30 queries is summarized in Figure 11.7(a & b). We find that DynaMem with both VLM-feature based and mLLM-QA based queries have a



(a) Failure modes on trials with differing query methods, VoxelMap (left) and multimodal LLM (right) in lab environments.



(b) Comparison with a static baseline in lab environments



(c) Failure modes in real home environments

Figure 11.7: Statistics of failure, broken down by failure modes, in our real robot experiments in the lab and in home environments. Statistics are collected over three environments and 30 open-vocabulary pick-and-drop queries for the lab experiments, and two environments and 17 pick-and-drop queries for the home environments, on objects whose locations change over time.

total success rate of 70%. This is a significant improvement over the OK-Robot system, which has a total success rate of 30%. Notably, DynaMem is particularly adept at handling dynamic objects in the environment: only 6.7% of the trials failed due to our system not being able to navigate to such dynamic objects in the scene. This is in contrast to the OK-Robot system, where 53.3% of the trials failed because it could not find an object that moved in the environment. In contrast, navigating to static goals fails in only 10% of the cases for DynaMem with VLM-feature, 13.3% for OK-Robot and 20% for DynaMem with mLLM-QA.

We observe that a common localization failure for VLM-feature based queries is that those features often act like a bag-of-words model; for example they may find a blue bowl when queried for

a red bowl. On the other hand, mLLM-QA based queries often fail because the objects are not present in the short context window, or the images in the context window are outdated. Since their failure modes are complementary, we hope to combine the two methods in future work to improve the overall performance.

11.4.2 Deployment in Home Environments

To understand the applicability of our system beyond lab environments that emulate real living space, we deployed DynaMem to two apartments in New York City. We ran experiments with DynaMem on a one one-room and one two-room environment. We ran a total of 17 long-horizon trials to understand the possible failure modes when deployed on a real scene. Since we observe complementary cases of failure from the two querying methods (Section 11.3.3.3) in our lab experiments in Section 11.4.1.1, we run these experiments with the hybrid querying method, setting the number of images returned by VoxelMap to be processed by mLLMs to be $k = 3$.

We found a total of 9 successes in our 17 real home trials, but it is significantly more interesting for us to understand how these systems may fail in real environments, as shown in Figure 11.7(c). We first observe that out of the 8 failures, 4 happened due to poor navigation, 3 happened due to our manipulation skills failing, and 1 was an error in placing the object at the target. Out of the navigation failures, half happened because of the target object was not found or was confused with another object, and the other half happened due to navigating to a suboptimal location for manipulation.

11.4.3 Ablations on an Offline Benchmark

Running real robot OVMM experiments can be expensive and time-consuming. So, we developed an offline benchmark called DynaBench to easily evaluate dynamic 3D visual grounding algorithms

on dynamic environments and perform algorithmic ablations. The benchmark isolates the query-response part of the dynamic semantic memory without robot navigation, exploration, and manipulation.

11.4.3.1 Data Collection

In the real world, the robot collects its own map-building data by exploring the environments. Following this, we collect the robot's runtime sensor data from three environments. To further enrich our benchmark, we simulate this process by taking posed RGB-D images on an iPhone Pro in six more environments. In all cases, we emulate environment dynamics by moving objects and obstacle locations in three successive rounds.

11.4.3.2 Data Labeling and Evaluation

We manually annotate queries and responses in the dataset. Each query has an associated natural language label q, object location $\vec{X} = (x, y, z)$, and an object radius ϵ. Since the environment is dynamic, each query also has an associated time t. For evaluation, at time t (i.e. after the memory algorithm has observed all the input data points with timestamp $< t$), we query the model with q. If it predicts an object location $\vec{X'} = (x', y', z')$, it's a success if $||X - X'||_2 \leqslant \epsilon$ and a failure otherwise. Since the robot may also encounter queries for objects it has not observed yet, we emulate negative queries by adding queries for objects (a) that have not been observed yet, or (b) that have been observed but were subsequently removed. For both of these query types, the model must respond with *not found*; otherwise it's counted as a failure.

11.4.3.3 Evaluation Results

Using our offline benchmark, we ablate design decisions of DynaMem as discussed in Section 11.3. Among these design decisions, the primary are: using feature embedding-based vs. mLLM-QA

Table 11.1: Ablating the design choices for our query methods for DynaMem on the offline DynaBench benchmark. We also present results from five human participants to ground the performances.

Query type	Variant	Success rate
Human	(average over five participants)	**81.9%**
VLM-feature	default (adding and removing points)	**70.6%**
	only adding points	67.8%
	no OWL-v2 cross-check	59.2%
	no similarity thresholding	66.8%
mLLM-QA	default (Gemini Pro 1.5)	**67.3%**
	Gemini Pro 1.5, no voxelmap filtering	66.8%
	Gemini Flash 1.5	63.5%
Hybrid	VLM-feature $\longrightarrow$ mLLM ($k = 3$)	**74.5%**

based language grounding, ablating components such as point removal or abstentiation from the algorithm, and trying different mLLMs. Due to API costs, we only evaluate Gemini models on the benchmark. We present our results in Table 11.1.

We see that performance of VLM-features and mLLM-QA follows the same order in the real world in the benchmark, corroborating the benchmark design. The best design choices are to both add and remove points, and to cross check with OWL-v2 on top of similarity thresholding for VLM-feature based grounding. For mLLM-QA based grounding, Gemini Pro outperforms Gemini Flash, and voxelmap based image filtering benefits the method. Moreover, we see that the hybrid method that uses VoxelMap feature filtering and then sends the top 3 images to the mLLM performs better than either method individually.

11.5 Limitations

In this work, we introduced DynaMem, a spatio-semantic memory for open-vocabulary mobile manipulation that can handle changes to the environment during operation. We showed in three real world environments that DynaMem can navigate to, pick, and drop objects even while object

and obstacle locations are changing. In the future, we could improve DynaMem performance by merging the VLM-feature queries and mLLM-QA queries, as they show complementary failure cases. Similarly, reasoning over both object and voxel level abstraction could speed up environment update when objects move. Our current system experiences a large number of manipulation failures: using mLLMs to detect and recover from failures [Etukuru et al. 2024] may increase the performance of the overall system. Finally, integrating with more skills, such as searching in cabinets or drawers, would improve the applicability of such OVMM systems in the real world.

Postscript

DynaMem closes a long-standing gap in the spatio-semantic memory architectures by creating a system that can update online in real time as the world around it changes. While critical, this is an incredibly hard system to manage – since there are many corner cases that can pop up in practice. Looking forward, there should be possible upgrades in how this system behaves – primarily by eliminating the hand-crafted heuristics like ray-casting and object clustering that we rely on when we update the state of the world. End-to-end learned systems may sound promising, but the truth is we still do not have the right kind of datasets that are needed to create the long-horizon intelligence critical for tackling real homes for days or weeks. The second important factor is making sure to study homes and environments that are truly lived in – diverse factors show up that make any of these environments dramatically different than what we can generate and simulate while sitting in a lab. Therefore, we also need to build systems that are holistic and can survive contact with the real world. Finally, running experiments like these can be costly – so there should be more research into minimal reproduction that are still able to distinguish between what methods work and don't work in the real world. Such a test bed would be invaluable pushing long-horizon mobile manipulation research forward.

Acknowledgements

This work was led by Peiqi Liu, co-authored with Zhanqiu Guo, Mohit Warke, Soumith Chintala, and Chris Paxton, and co-advised with Lerrel Pinto.

12 | Discussion

This thesis has presented a body of work focused on developing general robot intelligent that can solve a variety of problems in arbitrary human environments right out of the box. In Part I, we showed how we can learn representations for both the perceptions and the actions to make policy learning algorithms that are more scalable than their predecessors. Particularly, Chapter 2 showed the advantage of using self-supervised methods for pre-training a visual representation for a robot policy, a practice that we carried for the rest of this thesis. Then, in Chapters 3 to 5, we discussed two primary ideas: using a sequence of observations, rather than a single one, to get around non-markovian behavior of environments, and figuring out a hybrid discrete-continuous representation for actions that allows us to learn multi-modal behavior distributions. While these algorithms have their own limitations, especially in reconstructing the original precise actions in its full fidelity, they also properties of scalability that comes in handy in later chapters and will be helpful for future practitioners.

In Part II, we start taking a holistic view of robot learning in the wild and start building integrated systems, encompassing everything from physical hardware to datasets and learning algorithms. The benefit of taking a holistic view is that such approaches let us build, from scratch, robot systems that can solve novel tasks in novel home environments with few or zero shot approaches. In Chapter 6, we started by creating a hand-held data collection tool, a home pretraining dataset, and a learning algorithm that can learn new short horizon tasks in new environments with only 5

minutes of data and 15 minutes of fine-tuning a pre-trained model. Continuing on this path, we show in Chapter 7 that pooling or collecting in the wild data from a diversity of environments is then able to generalize to completely new environments and new object instances right out of the box, even on a completely new robot. With this system, for the first time we were able to create an end to end policy that one can simply download and run on a new robot in a new environment out of the box. As we were able to solve simpler problems, to create a path towards more complex challenges within the limits of academia, we created an open source bimanual mobile manipulator in Chapter 8. Once again, the systems we created are limited in the extent of long-horizon or dexterous behavior they can exhibit, but they pave the way towards building robot policies that can generalize to any human environment without having to fine-tune further on new data.

While the previous parts of this thesis focus on learning a short-horizon behavior policy for one-off behaviors, in Part III, we focus on creating robots capable of long-horizon mobile manipulation. In Chapter 9, we show how we can create a neural data structure for spatio-semantic robot memories by using representations from pre-trained large vision and language models that can resolve queries given in natural language in completely new environments zero shot. Armed with such a memory representation, Chapter 10 showed that we can decompose multi-step manipulation problems, and then combine the memory representation with pre-trained manipulation policies or heuristic to perform long-horizon mobile manipulation in novel home environments. However, one clear limitation of this method was its reliance on static environment, and we rid ourselves of it in Chapter 11 by creating an online, dynamic memory representation that can update with the environment changing over time. Despite the lengthened processing time of a query due to the prevalence of outdated memory, this method is able to perform long horizon mobile manipulation in a changing world due to the actions of the robot or a co-existing human. Despite their limitation on how quickly they can update their memory, or how quickly the robot can explore its environment to build the memory for the first time, they unlock many potential avenues for creating long-horizon autonomous behaviors in unstructured environments.

We are starting to barely scratch the surface of the complex problem of creating general intelligence for robots that can take over our tedious chores in homes, bodegas, fast-food restaurants, and factories. The work presented in this thesis is merely a prelude to that goal. The primary gap between what we have achieved here, and what we need to achieve to get to fully autonomous robot butler, lies in three primary direction: robust, error correcting behaviors; complex, dexterous task completion, and finally, autonomous, long horizon behavior on a much longer scale.

We have started our efforts in training general policies that work in arbitrary environments, but today the success probability of a single trial of such methods range between 80% [Shafiullah et al. 2023b] and 90% [Etukuru et al. 2024]. Even with only 10 such subtasks running in a sequence to accomplish a larger task, the success rate falls down to a measly 34%. Therefore, if we are to truly deploy such learned systems in the real world, our robots must have sub-task success rates which are much higher than what they are today, and have build in robust recovery behavior that will help them correct their own errors. There are many ways to build upon the work presented in this thesis to create such robust recovery behavior. The primary way would be to use some sort of reinforcement learning based fine-tuning of the pre-trained policies [DeepSeek-AI et al. 2025; Lambert et al. 2025] in the real world. Especially in the large language modeling world, it has been shown that models that already have a desired behavior can be "sharpened" to exhibit such behaviors more consistently. Therefore, the most important property for such a general model would be to have a non-zero success rate for the target task in any potential environment, and bootstrap the desired behavior from there using RL. Another potential direction for error detection and recovery behavior lies in world modeling – models that can foresee potential failures can also use planning to avoid such failures, or at least fail gracefully and ask for help. However, due to the intrinsic difficulty of world modeling, this path is harder to follow. We believe collecting rich and diverse data in the wild *en masse* with cheap setups as presented in Part II can give us a way forward to learn such diverse yet robust world models.

To achieve the level of dexterity of even a schoolchild seems incredibly far away for our robots. However, we believe the situation is set to improve soon due a new generation of cheap, precise, and dexterous robot hardware resulting from current wave of interest in robotics. With this new generation of robot hardware, it will be much easier to develop algorithms and policies that exhibit precise real world behavior, and improve the behavior precision by using hardware-intensive approaches such as real-world RL. With robust and reasonably cheap hardware, dexterity, especially of the multi-fingered variety, can walk the same path previously laid out by quadrupedal locomotion policies. To achieve fluid and reactive behavior that is characteristic of humans, however, we will need more than cheap, small, and precise motors – namely good sensors that fit everywhere and work robustly over time. Additionally, we will need to develop robot policies that rely less on vision, which is a cheap and ubiquitous sensing module, and rely more on other perception modules such as touch. Recent work in developing cheap and reproducible tactile sensors such as [Pattabiraman et al. 2025; Bhirangi et al. 2024] is a step forward in achieving this goal. The open question, however, remains of how to train dexterous policies that generalize outside of their training environments – since it is currently almost impossible to simulate diverse environments matching real world diversity, and data collection approaches in diverse real world setups may not scale directly to the multi-fingered hand case. Given this dilemma, we may have to invent additional frameworks that bridge the data collection and data generation mindsets from imitation learning and reinforcement learning.

However impressive demonstrations are possible today with end-to-end learning of robot policies, the question of how to enable them to act autonomously to complete a variety of general tasks over hours, days, or months still remains open. It is clear to see that teleoperating the robots over a very long period is prohibitively expensive, and so to bootstrap we may need a modular approach. The jury is still out, however, in finding out how to modularize them. This thesis, in Part III shows one way of doing so: with a flexible memory data structure that relies on certain heuristics to update. But should we instead be using end-to-end models for this? At what scale

would they become more useful, and what priors should we include in them? Memory is an hard and under-explored problem, not just in robotics, but in entirety of artificial intelligence, but there is a lot of nice regularizing properties of robotic memory that makes it a suitably concise subject of study. Another directions that can be built upon the work presented here is to find a more organic way of decomposing larger tasks into subtasks that does not depend on the intermediary of language and can operate on a lower level of abstraction. Finally, finding ways of incorporating human priors into robotic memory could be an interesting and largely beneficial topic of study – when a robot's environment changes, it does not happen adversarially or even randomly. Rather, there is a very strong prior over the distribution of new world states conditioned on the agents, both human and robotic, present in that scene. Therefore, by learning human patterns, robotic memories can become much more efficient at both responding to queries and updating its internal representation.

Beyond every proposed topic above, we must remember that the study of robotics, especially in human environments, should be in service of humanity. As we build better, stronger, and more capable robots, we must check in with ourselves to ensure that our robots are able to provide the value for humans that they promise. With the increase in capacity of robots, now is a great time to invest in understanding the interaction of robots and humanity to maximize the potential positive outcomes while minimizing the chance for intentional or unintentional harm.

This thesis has presented some work that start to pave the path towards generally intelligent robots that work in our service, out of the box, everywhere, but there is still a lot of work that remains. Towards that end, the methods and systems created in work will be helpful – and achieving that goal eventually will be their ultimate sign of success.

APPENDIX A

Appendix for Visual Imitation with Nearest Neighbors

A.1 VINN Pytorch Pseudocode

```
def dist_metric(x,y):
    return(torch.norm(x-y).item())


def calculate_action(dist_list,k):
    action = torch.tensor([0.0,0.0,0.0])
    top_k_weights = torch.zeros((k,))
    for i in range(k):
        top_k_weights[i] = dist_list[i][0]
    top_k_weights = softmax(-1*top_k_weights)
    for i in range(k):
        action = torch.add(top_k_weights[i]
            * dist_list[i][1], action)
```

```
    return(action)


def calculate_nearest_neighbors(query_img, dataset, k):
    query_embedding = encoder(query_img)
    for dataset_index in range(len(dataset)):
        dataset_embedding, dataset_translation = \
            dataset[dataset_index]
        distance = dist_metric(
                    query_embedding,
                    dataset_embedding
                )
        dist_list.append(
            (distance, dataset_translation, dataset_path)
        )
    dist_list = sorted(dist_list, key = lambda tup: tup[0])
    pred_action = calculate_action(dist_list, k)
    return pred_action
```

A.2 Network Architectures and Training Details

In this section, we will go over our implementation, network architectures, and training details for our various baselines.

Random Action We sampled a 3-d vector from $[-1, 1]^3$, normalized it, and used it as our action for this baseline.

Open Loop We computed the average action at frame t over all demonstrations from our dataset for this baseline for each frame number t.

Behavioral Cloning (end to end or from representations) For our parameterized model experiments, our encoding network is always a ResNet50, and our translation neural network is a three-layer MLP whose layer dimensions are 2048, 1024, 3. Our gripper model is a linear layer that predicts four gripper states from the encoder network output. We train BC-rep's MLPfor 8000 epochs with a learning rate of 0.001 using the Adam optimizer, and we train BC end-to-end for 100 epochs. On an RTX8000, given the learned representations, it takes 12 minutes on the Door Opening dataset to train both MLPs for BC from representations. For training the BC end to end model until convergence, it takes us three hours in total.

Implicit Behavioral Cloning (IBC) We used the official Github repo for Implicit Behavioral Cloning [Florence et al. 2022] offline experiments. We modified their Push from Pixels task to fit 3-d vectors bounded within $[-1, 1]^3$. Unfortunately, we could not use the space of normal vectors since the current published version of the IBC code does not support constrained action spaces.

We trained the standard Dense ResNet model provided with the IBC repo for encoders, and IBC-with-DFO framework for sampling actions. It took us about 6 hours to train the models end-to-end on our datasets on an RTX 8000 GPU for 10,000 steps. For every hyperparameter, we use the defaults for the learning to push from pixels task that is included in the IBC repository.

We computed the MSE loss from this model by first sampling 256 actions with DFO optimization, as it's done in IBC, and choosing the action with the highest assigned value out of it.

VINN For our BYOL-trained encoding network, we use a ResNet50 architecture, with the final linear for ImageNet classification replaced with an identity layer. We use the representation vector of size 2048. We fine-tune this network using BYOL for a 100 epochs on our demonstration

datasets with the ADAM optimizer and a 3×10^{-4} learning rate. To train this BYOL on the Door Opening dataset for 100 epochs it took approximately 3.5 hours on a workstation with one Nvidia RTX8000.

A.3 Robot details

We run all of our robots in the Hello Robot's Stretch [Kemp et al. 2022]. This robot has a dexterous wrist with 3 DoF, a telescopic 1 DoF arm on which it is mounted, and an 1-DoF lift on which the arm is mounted. The base of the robot is also capable of rotation and lateral motion, which gives the robot's end-effector a full 6-DoF capability.

On each step, the translation model predicts $\Delta(x, y, z)$ for the gripper, which is converted to the movement in the robot's joints with an inverse kinematics model. This model takes into account simpler objectives like avoiding self-collisions, but does not model avoiding issues like environment collisions.

For the robot observations, we use a standard webcam mounted on top of the robot wrist using a custom 3-d printed mount. The image captured by the robot is streamed over the network to a machine running the VINN algorithm, which responds with the predicted robot action.

A.4 Demonstration Collection Details

We use the DemoAT [Young et al. 2020] framework for collecting our demonstrations. We use a simple reacher-grabber tool availabe at hardware shops or online, fitted with a GoPro camera to do capture our observation frames. An image of this is shown in Fig. A.2,

We replaced the pads at the end of the robot gripper with simple 3-d printed nubs for easy resets of the robot, and we do the same on the reacher-grabber tool, as seen in Fig. A.3.

Figure A.1: Hello Robot's Stretch [Kemp et al. 2022], the robot model used in our experiments

To get visual observations, we mount a GoPro on top of the reacher grabber tool with a custom 3-d printed mount. We linearize the GoPro video in post-processing using `ffmpeg` to get rid of the wide-angle distortions, and extract the frames at one frame per second speed. Finally, using the extracted frames and the OpenSfM library, we reconstruct the 3-d movements between frames. We take the delta position changes between consecutive frames, and normalize them to get our

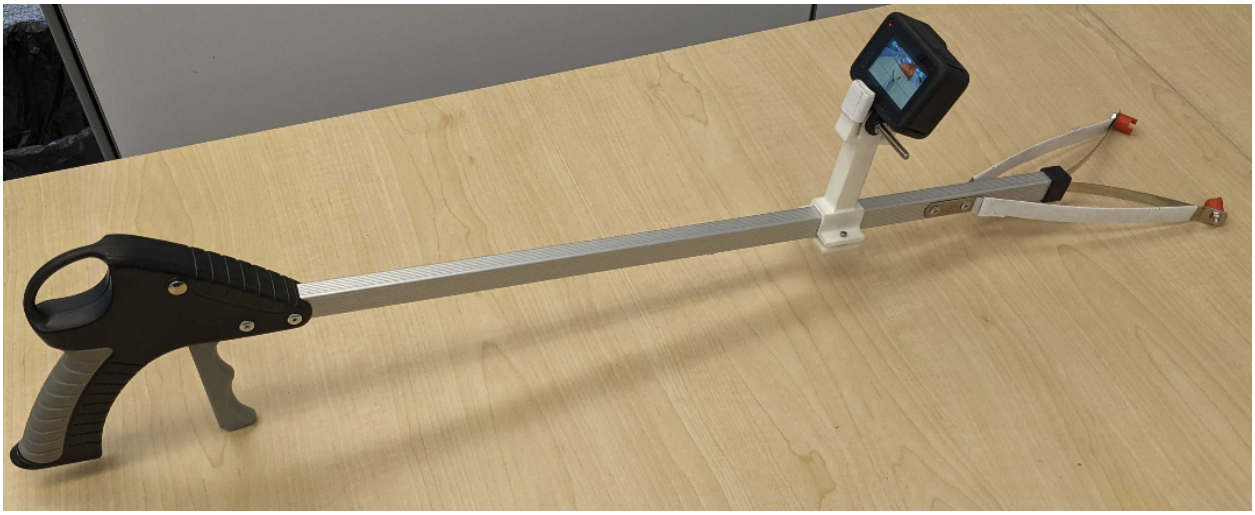


Figure A.2: Reacher grabber tool used for our demonstrations.

Figure A.3: Modified grip on the robot and the reacher grabber.

actions.

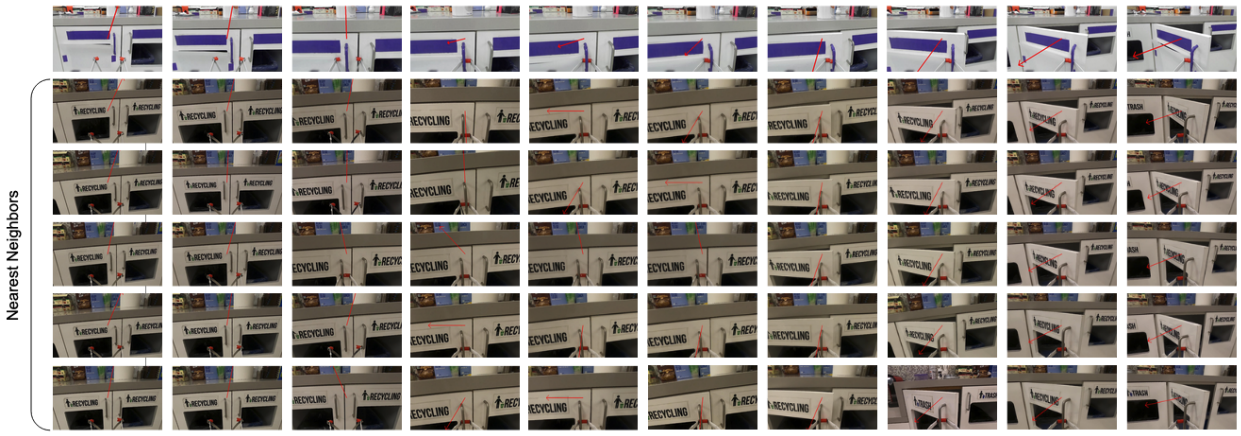


Figure A.4: The top row contains one rollout of VINN on a visually modified cabinet, under each image is the top 5 nearest neighbors from our demonstrations with the top one being the closest

APPENDIX B

Appendix for Behavior Transformers

Appendix

Diverse, multi-modal behaviors generated by our models on different environment are best experienced and understood in a video. We invite you to visit https://mahis.life/bet to see BeT models in action.

B.1 Environment and Dataset Details

Point mass environments: In the point mass environment, we have a simple point-mass agent with two-dimensional observation and action spaces. The observation of the agent denotes the (x, y) position of the agent, while the action sets the immediate $(\Delta x, \Delta y)$ displacement of the agent in the next timestep.

To show the effects of unimodal and multimodal behavioral cloning algorithms more cleanly, we also add a "snapping" effect to the environment which moves the agent close to the nearest integer coordinates after each step.

We generate random trajectories for each of our Multipath experiment datasets.

1. In the first one (Fig. 3.2), our dataset has two modes, which are colored differently in the figure based on the path taken at the fork.
   (a) In the first set of demonstrations, the point mass follows the trajectory $(1, 2)$, $(2, 2)$, $(2, 3)$, $(2, 4)$, $(3, 4)$, $(4, 4)$, $(4, 3)$, $(4, 2)$, $(5, 2)$.
   (b) In the second set of demonstrations, the point mass follows $(1, 2)$, $(2, 2)$, $(2, 1)$, $(2, 0)$, $(3, 0)$, $(4, 0)$, $(4, 1)$, $(4, 2)$, $(5, 2)$.
2. For the second Multipath environment (Fig. 3.5), there are three modes of demonstration, which are colored in the figure according to their first step direction.
   (a) In the first set of demonstration, the point mass follows $x = y$ from $(0, 0)$ to $(8, 8)$ with $\sqrt{2}$ size step increments.
   (b) In the second set of demonstration, the point mass follows straight lines from $(0, 0) \to (0, 4) \to (4, 4) \to (8, 4) \to (8, 8)$ with step size 1.
   (c) In the third set of demonstration, the point mass follows straight lines from $(0, 0) \to (4, 0) \to (4, 4) \to (4, 8) \to (8, 8)$ with step size 1.

CARLA ENVIRONMENT: We use the CARLA [Dosovitskiy et al. 2017] self-driving environment to examine BeT performance in environments with high-dimensional observation spaces. CARLA uses the Unreal Engine to provide a photo-realistic driving simulation. We create our environment on the Town04 map in CARLA 0.9.13. The observation space is $224 \times 224 \times 3$ RGB images from the vehicle, which are processed by an ImageNet-pretrained, frozen ResNet-18 to a 512-dimensional real-valued vector. The action space is $[-1, 1]^2$ with an accelerator-brake axis and a steering axis.

The dataset on this environment is collected with the built-in PID agent with minor tuning. We fix waypoints in the trajectory that the demonstration agent needs to follow. The waypoints

fork around two central blocks: one set of trajectories thus go to the left, while another set of demonstration trajectories go to the right. While collecting the demonstrations, we add some noise in the environment before executing an action so that there is some variation in the set of 100 total demonstrations that we collect in the environment.

We do not introduce any traffic participants in this environment intentionally as we intend to show the effects of cleanly bi-modal distributions on the learning algorithms in an environment more complicated than the point-mass environments.

Block-push environment: We use a simulated environment similar to Multimodal Push environment described in [Florence et al. 2022]. We take the environment implementation directly from the PyBullet [Coumans and Bai 2016] based implementation provided by Florence et al. [2022] in https://github.com/google-research/ibc/tree/master/environments.

In our environment, an XArm robot is situated in front of two blocks in a 0.75×1 plane. On the plane there are also two square targets. The goal of the agent is to push the blocks inside of the squares. However, the exact order of the block being pushed, or the combination of which block is pushed in which square doesn't matter. A block is considered successfully pushed if the center of the block is less than 0.05 away from a square.

On initialization, the blocks' positions are randomly shifted within a rectangle of side lengths $(0.2, 0.3)$, while the squares are randomly shifted within a rectangle of size $(0.01, 0.015)$. Additionally, the blocks were rotated at an uniformly arbitrary angle, while the target squares were rotated at an angle between $(\frac{\pi}{6}, -\frac{\pi}{6})$.

The demonstrations in this environments were collected with a hard-coded controller. There are two modes of multimodality inherent in the controller generated demonstartions. The controller:

1. Selects a block to start pushing first,

2. At the same time, independently chooses a target for the block to be pushed into.

3. Once the first block is pushed to a target, it pushes the second block to the remaining target.

Thus combinatorially, the controller is capable of four different modes of behavior. There are additional stochasticity in the controller behavior since there are many ways of pushing the same block into the same target.

The controller pushes the blocks to their targets following specific behavior primitives, such as moving to origin position, moving to a place collinear with a block and its target, and making a straight motion from that position towards the target unless the block rotates too much from its starting position.

Our models were trained on 1,000 demonstrations, all generated from the controller under the above randomized modes.

FRANKA KITCHEN ENVIRONMENT: For the final set of experiments, we use the Franka Kitchen environment originally introduced in the Relay Policy Learning [Gupta et al. 2019] paper. In that paper, the authors introduce a virtual kitchen environment where human participants in VR manipulated seven different objects in the kitchen: one kettle, one microwave, one sliding door, one hinged door, one light switch, and two burners. In total, we use 566 demonstrations collected by the researchers in that paper, where in each demonstration episode, each participant performed four manipulation task specified by the researchers in advanced.

The manipulator agent in simulator is a Franka Emika Panda robot, which is controlled through a 9-dimensional action space controlling the robot's joint and end-effector position. The 60-dimensional observation space is split into two parts, the first 30 dimension contains information about the current position of the interesting factors in the environment, while the last 30 dimensions contain information about the goal of the demonstrator or the agent. Note that in our demonstrations and our environments, we zero out the last 30 dimensions in all cases since we

assume goal is not labelled in the demonstrations and is not specified in the unconditioned rollouts of the model.

One thing to note that, while the D4RL [Fu et al. 2020] paper also has three versions of the dataset, we chose to use the original version of the collected data from the Relay Policy Learning [Gupta et al. 2019] paper. That is because the relay policy learning dataset is not labeled with intended tasks of the participants or rewards, while the D4RL dataset is geared towards that.

B.2 Implementation Details and Hyperparameters

B.2.1 Baselines

Multi-layer Perceptron with MSE For our MLP with MSE baselines, we trained fully connected neural networks with optionally BatchNorm layers. In each of our environment, we varied the depth and the width of the MLPs to fit them best according to the bias-variance trade-off, while training them on 95% of the dataset and testing on the remaining 5% on the dataset in terms of MSE loss.

Nearest Neighbor Nearest Neighbor is conceptually the simplest baseline we show in this paper. During training, our Nearest Neighbor model simply stores all the (o, a) pairs. During test time, given a query observation, o, we find the observation o' with the minimum Euclidean distance to that in the representation space, and execute the associated action a' in the environment.

While it is a simple baseline, we show that it has a surprisingly high effectiveness in simple environments like CARLA, or dense environments like Kitchen where there is less of a chance in going OOD simply by executing seen actions. On the other hand, in environments like Block-push where the model needs to interpolate or extrapolate more, the NN model fails more.

k-Nearest Neighbor with Locally Weighted Regression A slightly more robust version of NN for regression problems, k-NN with locally weighted regression or LWR, is the next baseline we use. In this baseline, we take the k-nearest neighbors (in all our cases, 5) in the observation representation space, and take a weighted average of their associated actions. The weighting is based on the negative exponent of the distance, or namely, $\exp -||o - o'||$, as seen in [Pari et al. 2021]. This model is better than simple Nearest Neighbors in interpolations, and thus we see a higher success in the Kitchen environment.

Continuous Generative Model: VAE with Gaussian Prior Following prior works[Pertsch et al. 2021], we use variational auto-encoders (VAE) for encoding and decoding sequences of actions into a smaller latent space. The VAE here learns to compress a sequence of $T = 10$ actions into a single latent variable z of 10 dimensions. The hyperparameters for training the VAE has been taken directly from Pertsch et al. [2021].

Concurrently with training the VAE, we train a state-conditioned latent prior model that tries to predict $P(z \mid o)$. This latent generator produces a vector of μ and σ which is sampled to find latent z, and we feed a Gaussian distributed variable z back into the decoder network where the action sequence is reconstructed. For the current observation o_t, sequence of reconstructed actions $a_t, \cdots, a_{t+9}$ are performed in a simulated environment.

The design choices of this algorithm has been heavily inspired by [Pertsch et al. 2021]. Although this model shows promise in theory, we found in practice that unconditional rollout from this model is not very successful. We believe the shortcoming is a result of random sampling from the z space that does not take into account the recently executed actions, and using a single-mode Gaussian as the state prior similar to [Pertsch et al. 2021], and thus this baseline is only slightly better than the MLP-MSE model.

Continuous Generative Models: Normalizing Flow with and without Prior Similar to Singh et al. [2020], we use a Normalizing Flow [Dinh et al. 2016] based generative model. We follow the architectural choices and the hyperparameters from [Singh et al. 2020] in our baseline implementation.

Our observation-conditioned Flow model is trained on the distribution $P(a \mid o)$ to continuously transform it into an identity Gaussian distribution of the same dimensions as a. To find a better prior than simply an identity Gaussian, we also trained a prior model that generates μ, σ of a Gaussian distribution given the observation o. We found that the prior improves the quality of the rollouts, however slightly.

We believe the under-performance of these continuous generative approaches were based on two major problems. One is that they fail to take historical context in concern, and by being a continuous distribution, returned less likely actions that led to more rollouts going OOD. Second, they were designed with a focus of making RL approachable by compressing the action space, which requires having a prior that is not so strict. However, most of BeT's performance comes from having a strong prior over the actions, which is only augmented by the action offset prediction.

Implicit Behavioral Cloning Implicit Behavioral Cloning (IBC) [Florence et al. 2022] takes a different approach in behavioral cloning, where instead of learning a model $f(o) := a$, we learn an energy based model $E(o, a)$ where the intended action a at any observation is defined as $\arg\min_a E(o, a)$. While this suffers from all the classic issues of training an EBM, like higher sample complexity and higher complexity in sampling, IBC models have been shown to have higher success in learning multi-modal and discontinuous actions.

As a baseline, we use the official implementation provided in `https://github.com/google-research/ibc` For the CARLA environment, we use equivalent hyperparameters from the "pushing from pixels" hyperparameters. For the Block-pushing environment, we use the "pushing from states"

hyperparameters. Finally, for the Kitchen environment, we use the "D4RL kitchen" hyperparameters.

While IBC is our strongest baseline, in our experience it is also one that is quite easy to overfit to our datasets. As a result, we monitored test performance over the training and had to employ early stopping for both the CARLA and the Block-pushing tasks.

Trajectory Transformers Trajectory Transformers [Janner et al. 2021], especially the variant that is trained without any rewards only on states and actions from demonstrations, seem similar to our approach, there are a few crucial differences. While we agree that BeT and Trajectory Transformer based behavior cloning both use some type of discretization to fit demonstration datasets with a minGPT, we believe that is where the similarities end. The primary differences between the algorithms is in our design choices: namely what distributions they model, and consequently how they treat the observations. The differences are explained more thoroughly below.

- **Modeled distribution**: From a provided set of demonstrations, trajectory transformers model the joint distribution P(action, observations). On the other hand, BeT models the conditional distribution P(action | observations). Modeling the joint distribution requires MinGPT to model the forward dynamics of the environment, which can be arbitrarily difficult based on the environment.

- **Observation discretization**: Because trajectory transformers have to model the observations as well, it needs to discretize the observation space. As a result, TT cannot extend to high dimensional observational spaces, such as visual observations. This limitation is also acknowledged by the authors of Trajectory Transformers. BeT, on the other hand, does not model the observations and thus does not need to discretize them. Thus BeT can scale to arbitrarily high dimensional observations, as we show in the CARLA environment

experiments, where BeT learns behaviors from high dimensional visual observations.

- **Efficient historical encoding**: Trajectory transformer encodes each (state, action) pair into a total of $|S| + |A|$ input/output tokens, while BeT encodes them into one input/output token. On a base MinGPT implementation that means a $O((|S| + |A|)^2)$ efficiency gain for BeT, or for example 4761x less compute for the same historic context in the Kitchen environment.

As a baseline, we trained and rolled out Trajectory Transformer on the Kitchen environment. It failed to complete any tasks for unconditioned, greedy, or beam search rollouts. We would like to note that the Kitchen environment is more complicated than the MuJoCo environments (HalfCheetah, Hopper, Walker2d, and Ant) that the paper experimented on. At the same time, this environment has an order of magnitude fewer samples on the training set (10^6 vs. approximately 120k). We tried both our own implementation and the implementation from https://github.com/Howuhh/faster-trajectory-transformer with the recommended parameters for the AntMaze environment, which is the largest environment used by the authors.

B.2.2 Algorithm Details

Loss function details: In this paper, we use two loss functions that are inspired by practices in computer vision, in particular object detection. The first of them is the Focal loss [Lin et al. 2017], and the second one is the Multi-task loss [Girshick 2015].

The Focal loss is a simple modification over the cross entropy loss. While the normal cross entropy loss for binary classification can be thought of $\mathcal{L}_{ce}(p_t) = -\log(p_t)$, the Focal loss adds a term $(1 - p_t)^\gamma$ to this, to make the new loss

$$\mathcal{L}_{focal}(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

This loss has the interesting property that its gradient is more steep for smaller values of p_t, while

flatter for larger values of p_t. Thus, it penalizes and changes the model more for making errors in the low-probability classes, while is more lenient about making errors in the high probability classes. Using this error in the object detection world has helped with class imbalance between different classes, and here it helps BeT learn to predict different k-means from the dataset even if their appearance in the dataset is not completely balanced.

For the multi-task loss, we use the formulation

$$\text{MT-Loss}\left(\mathbf{a}, \left(\langle \hat{a}_i^{(j)} \rangle\right)_{j=1}^{k}\right) = \sum_{j=1}^{k} \mathbb{I}[\lfloor \mathbf{a} \rfloor = j] \cdot \|\langle \mathbf{a} \rangle - \langle \hat{a}^{(j)} \rangle\|_2^2$$

This helps us penalize only the offset for the ground truth class, thus making sure the MinGPT is not trying to predict the right action offset through all classes and instead only trying to predict the action offset through the right class.

In practice, we optimize the combined loss, $\mathcal{L}_{focal} + \alpha \mathcal{L}_{mt}$ while α is a hyperparameter that just makes sure at initialization the two losses are of the same order of magnitude.

Compute details: All of our code was run in a single NVIDIA RTX 3080 GPU for state-based environments and RTX 8000 for image-based environments.

Performance measurement details: We measured the performance reported in the Section 3.3.5 in an NVIDIA RTX 3080 machine with AMD Threadripper 5950x CPUs. We took the average over three runs to minimize inter-run variances, and measured wall-clock time to report in the paper.

In terms of raw computation time to determine one action from the observations, in the Kitchen environment, BeT took 2.8 ms, while IBC took 52 ms and MLP, as the fastest point of comparison, took 0.5 ms. On the same environment, a single step of Trajectory Transformer took 867.86 ms, on an implementation that used more advanced tricks such as attention caching.

Hyperparameters list: We present the BeT hyperparameters in Table B.1 below:

Table B.1: Environment-dependent hyperparameters in BeT.

Hyperparameter	Point-mass	CARLA	Block-push	Kitchen
Layers	1	3	4	6
Attention heads	2	4	4	6
Embedding width	20	256	72	120
Dropout probability	0.1	0.6	0.1	0.1
Context size	2	10	5	10
Training epochs	10	40	350	50
Batch size	64	128	64	64
Number of bins k	2; 3	32	24	64

However, we have found that as long as the model does not overfit, a wide range of parameters all yield favorable results for BeT; thus, this table should be taken as reference values for reproducing our results rather than the only parameter sets that work.

Apart from that, we have some hyperparameters that are shared across all BeT experiments. They are reproduced in Table B.2.

Table B.2: Shared hyperparameters for BeT training

Name	Value
Optimizer	Adam
Learning rate	1e-4
Weight decay	0.1
Betas	(0.9, 0.95)
Gradient clip norm	1.0

B.2.3 Pseudocode

See the pseudocode described on Algorithm 1.

B.2.4 Architecture and Implementation

For our implementation, we used the MinGPT [Karpathy 2020] repository almost as-is. We modified the input token conversion layer to a linear projection layer to handle our continuous, instead of discrete, inputs. Apart from that, we followed the MinGPT architecture quite exclusively, with successive attention layers with a number of attention head and embedding dimensions. Between the layers, we used dropout regularization same as [Karpathy 2020].

For the smallest tasks, like point-mass environments, we used models with approximately 10^4 parameters, which went up to around 10^6 for Kitchen environments.

B.3 Ablation studies

In this section, we provide more details about the ablation studies presented in the main paper, as well as present detailed plots of our ablation studies that compare different versions of the BeT architecture.

B.3.1 Ablating historical context

One of the reasons why we used transformer-based generative networks in our work is because of our hypothesis that having historical context helps our model learn better behavioral cloning. Our experiments are performed by using the same model and simply providing sequences of length one on training and test time. As we can see on Sec. 3.3.5, having some historical context helps our model learn much better.

B.3.2 Ablating the core model in the architecture

To ablate the core MinGPT transformer model in the architecture, we run three ablations, where we replace it respectively with a fully-connected multi-layer perceptron (MLP) network, a temporal convolution network, and an LSTM-based recurrent neural network.

Multi-Layer Perceptrons: Since generally MLP networks are not capable of taking in historical context in consideration, we instead stack the last t frames of observation to pass into the MLP network. Near the beginning of a trajectory, the stack of observation is zero-padded to t frames. For the intermediate layers in the MLP, we keep the same width and the number of layers as the corresponding MinGPT.

Temporal Convolution: Convolutions over the sequence length has been used in numerous prior works [Oord et al. 2016; Kalchbrenner et al. 2016; Dauphin et al. 2017; Gehring et al. 2017; Bai et al. 2018] for sequence modeling. As a baseline, we implement such temporal convolutional network to replace our MinGPT-based trunk. We perform a temporal convolution over the same period of history that is provided to our transformer models. We found that the performance of the temporal convolution models are constantly lower than our MinGPT based models. However, temporal convolutional networks are easier to fit on our data compared to RNNs.

LSTM-based RNN: Recurrent neural networks (RNNs) were the previous state-of-the-art for sequence modeling before transformer-based models. In this work, we compare against an Long-short term memory (LSTM) [Gers et al. 2000] based RNN instead of a transformer based trunk. We find that even with sufficient model capacity, the RNN based model took significantly longer than our MinGPT model to fit the same dataset. Moreover, the quality of fit was worse, both in training and test time. Finally, in open-ended rollouts, this performance downgrade is reflected in

a far lower success rate for completing tasks in the environment (Table. 3.3).

Algorithm 1: Learning Behavior Transformer from a dataset of behavior sequences.

Input: Dataset $(o_{t,i}, a_{t,i})_{t,i}$ for $0 \leqslant i \leqslant$ number of demonstrations, $0 \leqslant t \leqslant$ maximum episode lengths, intended number of clusters k and context history length h.
Initialize: θ_M the parameters for MinGPT, $\{A_i\}_{i=1}^{k}$ cluster centers randomly in the action space.

Learn k-means encoder/decoder:
Using all possible $a_{t,i}$, learn the k cluster centers using the k means algorithm.
Set $\{A_i\}_{i=1}^{k}$ as the learned cluster centers.
Define functions:
$\lfloor a \rfloor := \arg\min_{i=1}^{k} ||a - A_i||$
$\langle a \rangle := a - \lfloor a \rfloor$
$\text{Enc}(a) := (\lfloor a \rfloor, \langle a \rangle)$
$\text{Dec}(\lfloor a \rfloor, \langle a \rangle) := A_{\lfloor a \rfloor} + \langle a \rangle$
Train MinGPT trunk of BeT:
while *Not converged* **do**
  Sample trajectory subsequence $(o_t, a_t), \cdots, (o_{t+h-1}, a_{t+h-1})$ from the dataset.
  Feed in the observations $(o_t, o_{t+1}, \cdots, o_{t+h-1})$ into the MinGPT.
  Get categorical distribution probabilities $p_{\tau,c}$ for $t \leqslant \tau \leqslant t+h-1, 1 \leqslant c \leqslant k$.
  Compute focal loss $\mathcal{L}_{ce}$ of $p_{\tau,c}$ against ground truth class $\lfloor a_\tau \rfloor$, for all τ, c .
  Get the residual action offset per class, $\langle a_{\tau,c} \rangle$, for all τ, c from MinGPT.
  Calculate the multi-task loss, $\mathcal{L}_{mt}$, against true class predicted offset, $\sum_\tau ||\langle a_{\tau,\lfloor a_\tau \rfloor} \rangle - \langle a_\tau \rangle||_2^2$
  Backprop using the normalized loss, $\mathcal{L}_{ce} + \alpha \mathcal{L}_{mt}$ where α makes the losses of equal magnitude.
end while
Running on the environment:
while *Episode not completed* **do**
  Stack the last h observations in the environment, $(o_t, o_{t+1}, \cdots, o_{t+h-1})$ and feed into MinGPT.
  Get categorical probabilities $p_{\tau,c}$ for $t \leqslant \tau \leqslant t+h-1, 1 \leqslant c \leqslant k$ from the MinGPT.
  Sample a class c from $p_{t+h-1,c}$ for $1 \leqslant c \leqslant k$.
  Get the associated action offset, $\langle a_{t+h-1,c} \rangle$ from the MinGPT.
  Decode into full continuous action, $\overline{a}_{t+h-1} := \text{Dec}(c, \langle a_{t+h-1,c} \rangle)$
  Execute decoded action $\overline{a}_{t+h-1}$ into environment.
end while

APPENDIX C

Appendix for Conditional Behavior Transformers

Appendix

C.1 Behavior Transformers

We use Behavior Transformers from [Shafiullah et al. 2022] as our backbone architecture, building our conditional algorithm on top of it. In this section, we describe the BeT architecture and the training objective to help the readers understand the details of our algorithm.

C.1.1 BeT training objective

Given an observation o and its associated ground truth action a, we will now present the simplified version of how the BeT loss is calculated.

Let us assume the BeT model prediction is $\pi(o)_d \in \mathbb{R}^k, \pi(o)_c \in \mathbb{R}^{k \times |A|}$ for the discrete and the continuous parts of the predictions. Let us also assume that $\lfloor a \rfloor$ is the discrete bin out of the k

bins that a belongs to, and $\langle a \rangle = a - \text{BinCenter}(\lfloor a \rfloor)$. Then, the BeT loss becomes

$$\mathcal{L}_{\text{BeT}} = L_{focal}(\pi(o)_d, \lfloor a \rfloor) + \lambda \cdot L_{MT}(\langle a \rangle, \pi(o)_c)$$

Where L_{focal} is the Focal loss [Lin et al. 2017], a special case of the negative log likelihood loss defined as

$$\mathcal{L}_{focal}(p_t) = -(1 - p_t)^{\gamma} \log(p_t)$$

and L_{MT} is the multi-task loss [Girshick 2015] defined as

$$\text{MT-Loss}\left(\mathbf{a}, \left(\langle \hat{a}_i^{(j)} \rangle\right)_{j=1}^{k}\right) = \sum_{j=1}^{k} \mathbb{I}[\lfloor \mathbf{a} \rfloor = j] \cdot \|\langle \mathbf{a} \rangle - \langle \hat{a}^{(j)} \rangle\|_2^2$$

C.2 Implementation Details

C.2.1 Implementation used

In our work, we base our C-BeT implementation off of the official repo published at https://github.com/notmahi/bet. For the GCBC, WGCSL, and GoFAR baselines, we use the official repo released by the GoFAR authors https://github.com/JasonMa2016/GoFAR/.

C.2.2 Hyperparameters list:

We present the C-BeT hyperparameters in Table C.1 below, which were mostly using the default hyperparameters in the original [Shafiullah et al. 2022] paper:

The shared hyperparameters are in Table C.2.

Table C.1: Environment-dependent hyperparameters in BeT.

Hyperparameter	CARLA	Block-push	Kitchen
Layers	3	4	6
Attention heads	4	4	6
Embedding width	256	72	120
Dropout probability	0.6	0.1	0.1
Context size	10	5	10
Training epochs	40	350	50
Batch size	128	64	64
Number of bins k	32	24	64
Future conditional frames	10	3	10

Table C.2: Shared hyperparameters for BeT training

Name	Value
Optimizer	Adam
Learning rate	1e-4
Weight decay	0.1
Betas	(0.9, 0.95)
Gradient clip norm	1.0

C.3 Robot Environment Demonstration Trajectories

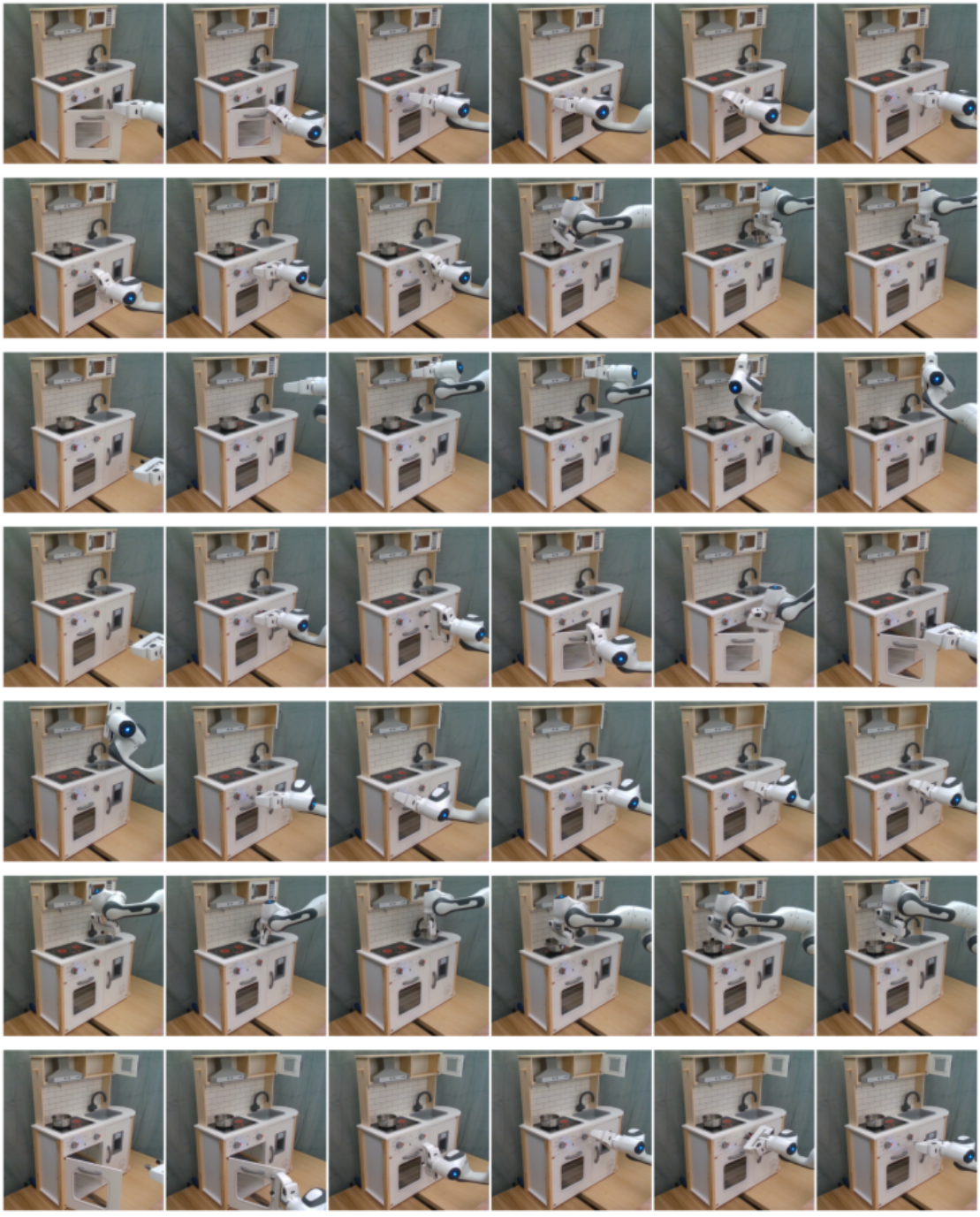


Figure C.1: Sample demonstration trajectories for the real kitchen environment.

C.4 Simulated Environment Rollout Trajectories

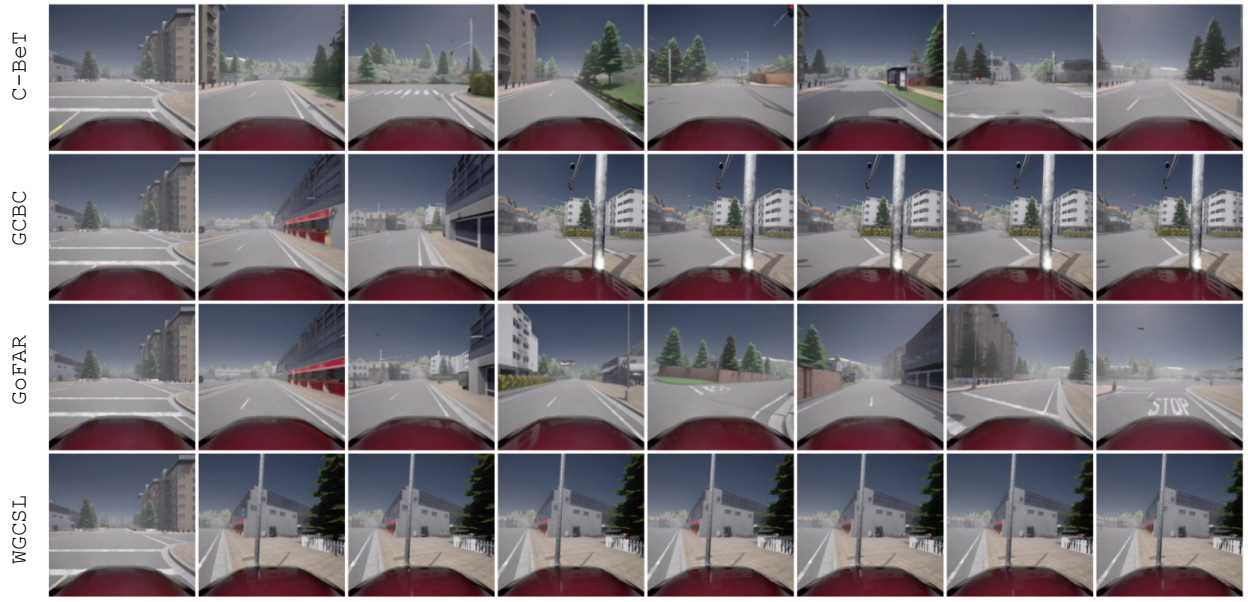


Figure C.2: Sample demonstration trajectories for the CARLA self driving environment, conditioning on going to the right path.

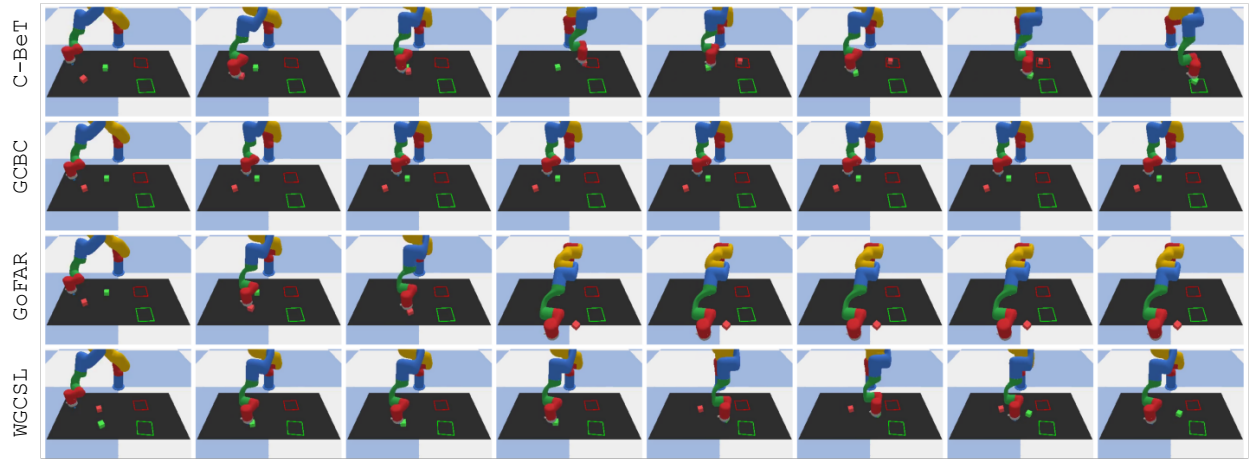


Figure C.3: Sample demonstration trajectories for the multi-modal block pushing environment, conditioning on pushing the green block to green square and red block to red square.

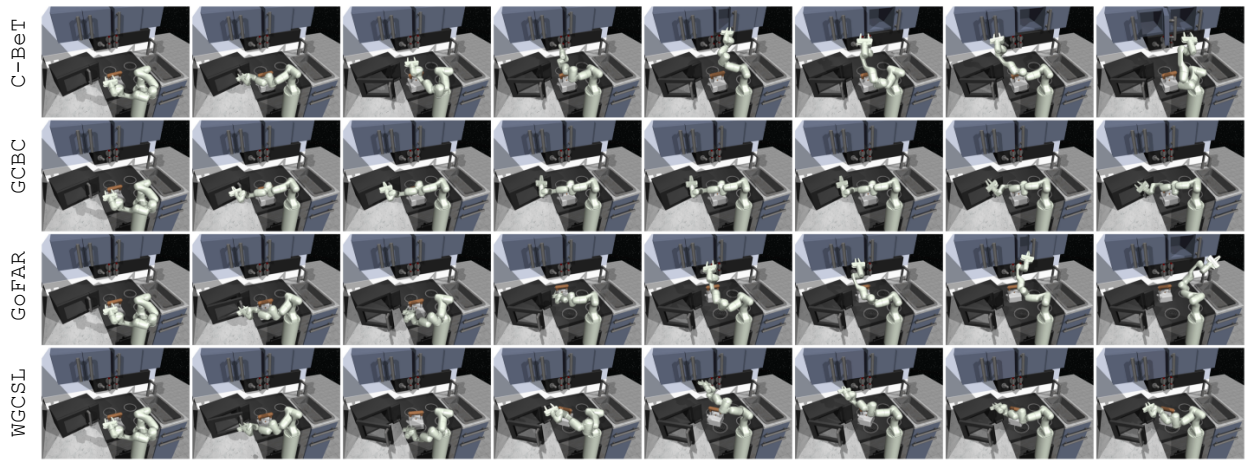


Figure C.4: Sample demonstration trajectories for the Franka Kitchen environment, conditioning on completing the microwave, bottom knob, slide cabinet, hinge cabinet tasks.

APPENDIX D

Appendix for Vector-Quantized Behavior Transformers

D.1 Experimental and Dataset

D.1.1 Simulated environments

Across our experiments, we use a variety of environments and datasets to evaluate VQ-BeT. We give a short descriptions of them here, and depiction of them in Figure 5.3:

- **Franka Kitchen:** We use the Franka Kitchen robotic manipulation environment introduced in [Gupta et al. 2019] with a Franka Panda arm with a 7 dimensional action space and 566 human collected demonstrations. This environment has seven possible tasks, and each trajectory completes a collection of four tasks in some order. While the original environment is state-based, we create an image-based variant of it by rendering the states with the MuJoCo renderer as an 112 by 112 image. In the conditional variant of the environment, the model is conditioned with future states or image goals (Image Kitchen).

- **PushT:** We adopt the PushT environment introduced in [Chi et al. 2023] where the goal is to push a T-shaped block on a table to a target position. The action space here is two-dimensional end-effector velocity control. Similar to the previous environment, we create an image based variant of the environment by rendering it, and a goal conditioned variant of the environment by conditioning the model with a final position. This dataset has 206 demonstrations collected by humans.
- **BlockPush:** The BlockPush environment was introduced by Florence et al. [2022] where the goal of the robot is to push two red and green blocks into two (red and green) target squares in either order. The conditional variant is conditioned by the target positions of the two blocks. The training dataset here consists of 1,000 trajectories, with an equal split between all four possibilities of (block target, push order) combinations, collected by a pre-programmed primitive.
- **UR3 BlockPush:** In this task, an UR3 robot tries to move two blocks to two goal circles on the other side of the table [Kim et al. 2022]. Each demonstration is multimodality, since either block can move first. In the non-conditional setting, we evaluate whether each block reaches the goal, while in the conditional setting, we evaluate in which order the blocks get to the given target point.
- **Multimodal Ant:** We adopt a locomotion task that requires the MuJoCo Ant [Brockman et al. 2016] robot to reach goals located at each corner of the map. The demonstration contains trajectories that reach the four goals in different orders. In the conditional setting, the performance is evaluated by reaching two goals given by the environment, while in the unconditional setting, the agent tries to reach all four goals.
- **nuScenes self-driving:** Finally, to evaluate VQ-BeT on environments beyond robotics, we use the nuScenes [Caesar et al. 2020] self-driving environment as a test setup. We use the preprocessed, object-centric dataset from Mao et al. [2023a] with 684 demonstration scenes where

the policy must predict the next six timesteps of the driving trajectory. In this environment, the trajectories are all goal-directed, where the goal of which direction to drive is given to the policy at rollout time. In Appendix Section D.3.2, we detail how we process the GPT-Driver Mao et al. [2023a] dataset for use in our method.

D.1.2 Real-world environments

We run our experiments on a kitchen-like environment, with a toaster oven, a mini-fridge, and a small can in front of them, as seen in Fig. 5.3. In this environment, we define the tasks as opening or closing the fridge or toaster, and moving the can from the table to the fridge or toaster and vice versa. During data collection and evaluation, the starting position for the gripper and the position of the cans are randomized within a predefined area, while the location of the fridge and the toaster stays fixed. We use a similar robot and data collection setup as Dobb·E [Shafiullah et al. 2023b], using the Stick to collect 45 demonstrations for each task, using 80% of them for training and 20% for validation, and using the Hello Robot: Stretch [Kemp et al. 2022] for policy rollouts. While some of the single tasks can only be completed in one way, the we also test the model on sequences of two tasks, for example closing oven and fridge, which can be completed in multiple ways. This task multi-modality is also captured in the dataset: tasks that can be completed in multiple ways have multi-modal demonstration data.

D.2 Additional Results

		C-BeT	C-BESO	CFG-BESO	**VQ-BeT**
Kitchen	Full	3.09	3.75	3.47	**3.78**
	1/4	2.77	2.62	3.07	**3.46**
	1/10	2.59	2.67	2.73	**2.95**
Image Kitchen	Full	2.41	2.00	1.59	**2.60**
Ant Multimodal	Full	1.68	1.14	0.92	**1.72**
	1/4	0.85	0.58	0.52	**1.23**
	1/10	0.35	0.39	0.40	**1.06**
BlockPush Multimodal	Full	0.87	**0.93**	0.88	0.87
	1/4	0.48	0.52	0.47	**0.62**
	1/10	0.10	**0.29**	0.17	0.13
UR3 Multimodal	$-\ell_1$	-0.129	-0.090	-0.091	**-0.085**
	p1	**1.00**	0.98	0.97	**1.00**
	p2	0.67	**0.96**	0.94	0.94
PushT	Final Coverage	0.02	0.30	0.25	**0.39**
	Max Coverage	0.11	0.41	0.38	**0.49**
Image PushT	Final Coverage	0.01	0.02	0.01	**0.10**
	Max Coverage	0.02	0.02	0.02	**0.12**

Table D.1: Quantitative results of VQ-BeT and related baselines on conditional tasks.

		BeT	DiffusionPolicy-C	DiffusionPolicy-T	**VQ-BeT**
PushT	Final Coverage	0.39	0.73	0.74	**0.78**
	Max Coverage	0.73	**0.86**	0.83	0.80
Image PushT	Final Coverage	0.01	0.66	0.45	**0.68**
	Max Coverage	0.01	**0.82**	0.71	0.73
Kitchen	p1	0.99	0.94	0.99	**1.00**
	p2	0.93	0.86	**0.98**	**0.98**
	p3	0.71	0.56	0.87	**0.91**
	p4	0.44	0.26	0.60	**0.77**
	p3-Entropy	**3.44**	3.18	3.38	3.42
	p4-Entropy	4.01	3.62	3.89	**4.07**
Image Kitchen	p1	0.97	0.99	0.97	**1.00**
	p2	0.73	**0.95**	0.90	0.93
	p3	0.51	0.73	**0.75**	0.67
	p4	0.27	**0.44**	0.39	0.38
	p3-Entropy	3.03	2.36	3.01	**3.20**
	p4-Entropy	2.77	2.93	**3.55**	3.32
Ant Multimodal	p1	0.91	**0.96**	0.87	0.94
	p2	0.79	0.81	0.78	**0.83**
	p3	0.67	0.73	0.69	**0.75**
	p4	0.36	0.62	0.56	**0.70**
	p3-Entropy	3.89	4.26	**4.27**	4.19
	p4-Entropy	3.55	4.18	4.11	**4.20**
BlockPush Multimodal	p1	0.96	0.36	**0.99**	0.96
	p2	0.71	0.11	**0.94**	0.83
	p2-Entropy	1.95	1.94	1.95	**1.99**
UR3 Multimodal	p1	0.84	**1.00**	**1.00**	**1.00**
	p2	0.75	0.83	0.82	**0.84**
	p2-Entropy	**0.99**	0.91	0.98	**0.99**

Table D.2: Quantitative results of VQ-BeT and related baselines on non-conditional tasks.

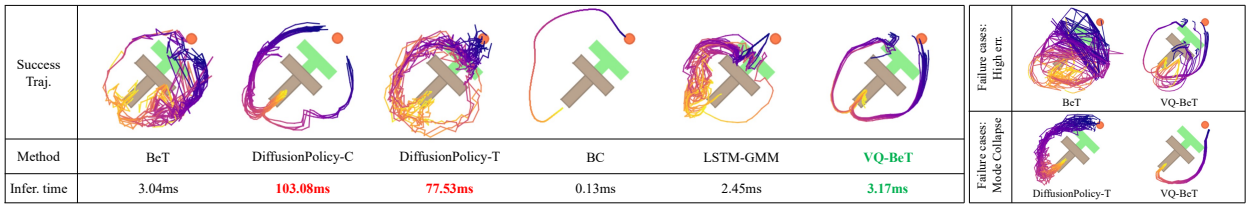


Figure D.1: Multi-modal behavior visualization on pushing a T-block to target. On the left, we can see trajectories generated by different algorithms and their inference time per single step, where VQ-BeT generate smooth trajectories to complete the task with both modes with short inference time. On the right, we can see failure cases of VQ-BeT and related baselines due to high error and mode collapse.

		L2 (↓)				Collision (%) (↓)			
		1s	2s	3s	Avg.	1s	2s	3s	Avg.
ST-P3 metrics	ST-P3 [Hu et al. 2022]	1.33	2.11	2.90	2.11	0.23	0.62	1.27	0.71
	VAD [Jiang et al. 2023]	0.17	0.34	**0.60**	**0.37**	0.07	0.10	0.24	0.14
	GPT-Driver [Mao et al. 2023a]	0.20	0.40	0.70	0.44	0.04	0.12	0.36	0.17
	Agent-Driver [Mao et al. 2023b]	**0.16**	0.34	0.61	**0.37**	**0.02**	**0.07**	**0.18**	**0.09**
	Diffusion-based Traj. Prediction	0.21	0.43	0.80	0.48	**0.01**	**0.07**	0.35	0.14
	VQ-BeT	0.17	**0.33**	**0.60**	**0.37**	**0.02**	0.11	0.34	0.16
UniAD metrics	NMP [Zeng et al. 2019]	-	-	2.31	-	-	-	1.92	-
	SA-NMP [Wei et al. 2021]	-	-	2.05	-	-	-	1.59	-
	FF [Hu et al. 2021]	0.55	1.20	2.54	1.43	0.06	0.17	1.07	0.43
	EO [Khurana et al. 2022]	0.67	1.36	2.78	1.60	0.04	0.09	0.88	0.33
	UniAD [Hu et al. 2023]	0.48	0.96	1.65	1.03	0.05	0.17	0.71	0.31
	GPT-Driver [Mao et al. 2023a]	0.27	0.74	1.52	0.84	0.07	0.15	1.10	0.44
	Agent-Driver [Mao et al. 2023b]	**0.22**	0.65	**1.34**	0.74	0.02	**0.13**	**0.48**	**0.21**
	Diffusion-based Traj. Prediction	0.27	0.78	1.83	0.96	**0.00**	0.27	1.21	0.49
	VQ-BeT	**0.22**	**0.62**	**1.34**	**0.73**	0.02	0.16	0.70	0.29

Table D.3: *(Lower is better)* Trajectory planning performance on the nuScenes [Caesar et al. 2020] self-driving environment. We **bold** the best performing model. Note that while Agent-Driver outperforms us in some Collision avoidance benchmarks, it is because they use a lot more information than what is available to our agent, namely the road lanes and the shoulders information, without which avoiding collision is difficult for our model or GPT-Driver [Mao et al. 2023a]. Even with such partial information about the environment, VQ-BeT can match or beat the SOTA models in predicting L2 distance from ground truth trajectory.

Control method	Close Toaster	Close Fridge	Can to Toaster	Can to Fridge	Can to Fridge → Close Fridge	Close Fridge and Toaster	Total
Closed loop ($n = 1$)	9/10	8/10	10/10	10/10	4/10	6/10	47/60
Receding horizon ($n = 3$)	0/5	0/5	0/5	0/5	0/5	0/5	0/30

Table D.4: Quantitative results of running diffusion policy [Chi et al. 2023] with closed-loop vs. receding horizon control in real-world robot experiments, where n is the number of actions executed at each timestep. We select four single-phase tasks and two two-phase tasks in which diffusion policy does well with closed-loop control, and compare with the same policy with receding horizon control by executing multiple predicted actions at each timestep. We see the diffusion policy with an action sequence executed per timestep goes out of distribution quite easily and fails to complete any tasks on this set of experiments.

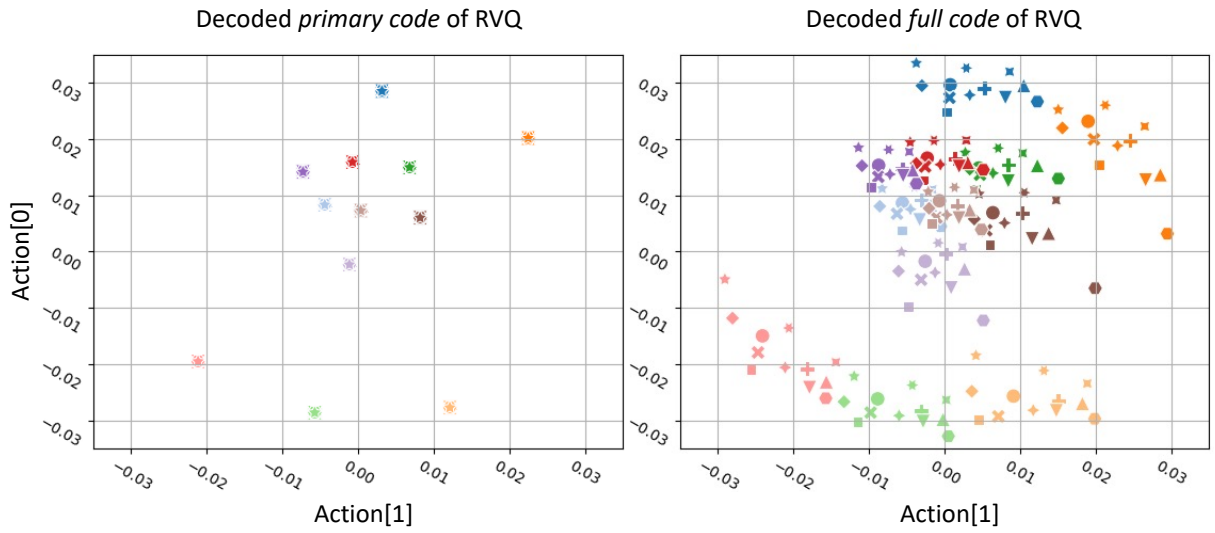


Figure D.2: Action centroids of primary codes and full combination of the codes. On the left, we represent centroids of the raw action data obtained by decoding (total of 12) primary codes learned from Blockpush Multimodal dataset. On the right, we show the decoded action of the centroids corresponding to all 144 possible combinations of full the codes. We can see that the primary codes, represented by different colors in each figure, are responsible for clustering in the coarse range, while full-code representation provides further finer-grained clusters with secondary codes.

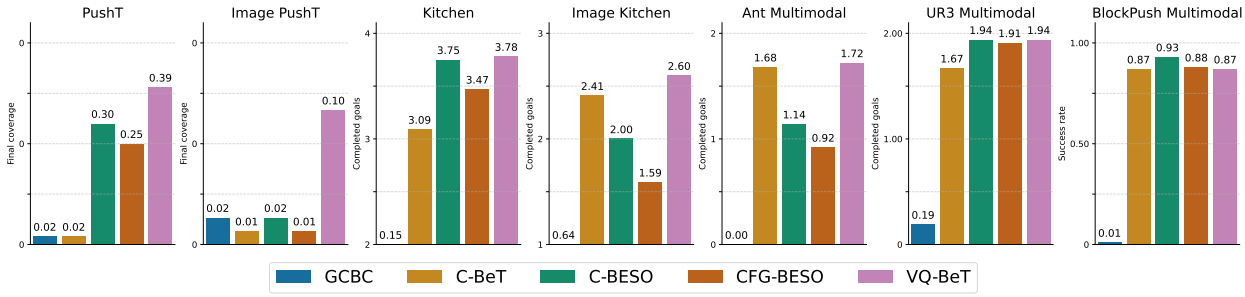


Figure D.3: Evaluation of conditional tasks in simulation environments of VQ-BeT and related baselines. VQ-BeT achieves the best performance in most simulation environments and comparable performance with the best baseline on BlockPush.

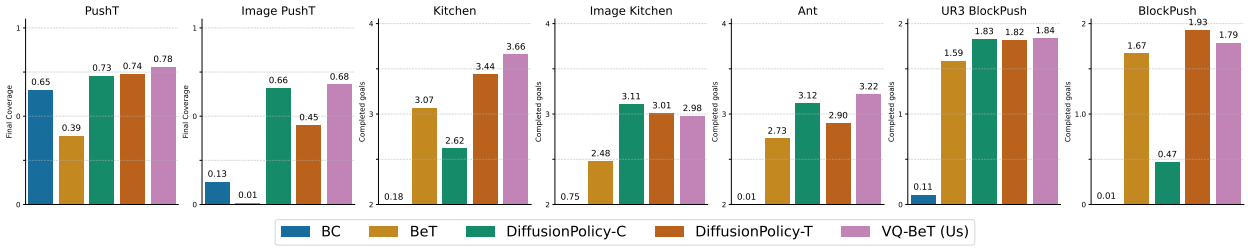


Figure D.4: Evaluation of unconditional tasks in simulation environments of VQ-BeT and related baselines. VQ-BeT achieves the best performance in most simulation environments and comparable performance with the best baseline on BlockPush and Image Kitchen.

D.2.1 VQ-BeT with larger Residual VQ Codebook

		Original codebook (Vanilla VQ-BeT)	Extended Codebook (Vanilla VQ-BeT)	Extended Codebook (VQ-BeT + Deadcode Masking)
Ant Multimodal (Unconditional)	Codebook Size	10	32	32
	# of Code Combinations	100	1024	1024
	$(\cdot/4)$	**3.22**	3.01	3.11
	p3-Entropy	4.19	4.23	**4.33**
	p4-Entropy	4.20	4.24	**4.32**
Ant Multimodal (Conditional)	Codebook Size	10	48	48
	# of Code Combinations	100	2304	2304
	$(\cdot/2)$	1.72	1.75	**1.81**
Kitchen (Unconditional)	Codebook Size	16	64	64
	# of Code Combinations	256	4096	4096
	$(\cdot/4)$	3.66	**3.75**	3.7
	p3-Entropy	**3.42**	3.01	3.10
	p4-Entropy	**4.07**	3.57	3.74
PushT (UnConditional)	Codebook Size	16	64	64
	# of Code Combinations	256	4096	4096
	Final Coverage	0.78	0.77	**0.79**
	Max Coverage	0.80	0.80	**0.82**
Kitchen (Conditional)	Codebook Size	16	256	256
	# of Code Combinations	256	65536	65536
	$(\cdot/4)$	**3.78**	3.61	3.56

Table D.5: Evaluation of conditional and unconditional tasks in simulation environments of VQ-BeT with extended size of Residual VQ codebook.

In this section, we present additional results to evaluate the performance of VQ-BeT with larger residual VQ codebooks. While the results of VQ-BeT across the manuscript were obtained using 8 to 16-sized codebooks, resulting in 64 to 256 code combinations (Table D.6), here, VQ-BET was trained on codebooks with 10 to 250 times more combinations, as detailed in Table D.5. First, we evaluate VQ-BeT with extended codebook size without any modifications ('Vanilla VQ-BeT'). Next, we test VQ-BeT with an additional technique where the code combinations that do not appear in the dataset are masked with a probability of zero at sampling time to eliminate the possibility of these combinations.

As shown in Table D.5, we find that increasing the number of combinations ($\times 10 \sim \times 250$) had little impact on performance in most environments. In environments Ant Multimodal (Conditional) and PushT (Unconditional), overall performance slightly increased as the size of the VQ codebook increased. In environments Ant Multimodal (Unconditional) and Kitchen (Unconditional), we

see that there is a performance and entropy trade-off as the size of the codebook increases. The only environment where the performance of VQ-BeT decreased with the extended size of the codebook was Kitchen (Conditional). Also, we see that there is no consistent evidence on whether using masking the deadcode (code combinations that do not appear in the dataset) is better: in Ant and PushT environments, masking led to similar or better performance, while in the Kitchen environment, we find similar or slightly worse performance with masking.

Overall, we conclude that VQ-BeT has robust performance to the size of the codebook if it is enough to capture the major modes in the dataset. We conjecture that this robustness is due to VQ-BeT assigning appropriate roles between primary and secondary codes as the codebook size increases. For example, in the Kitchen (Conditional) environment where we have increased the number of possible combinations by 256, the code prediction accuracy rate has decreased by only $\times 0.08$ of its original accuracy rate, while the primary code prediction retained $\times 0.8$ of its original accuracy rate. Interestingly, Despite this large difference, the performance difference between the two is small, around 4.5% (3.78 vs 3.61). These results suggest that VQ-BeT could rely on the resolution of the primary code in large VQ codebook size, while using less weight on the secondary code to handle the excessive number of code combinations, leading to robust performance to the size of the codebook.

D.3 Implementation Details

D.3.1 Model Design Choises

Hyperparameter	Kitchen	Ant	BlockPush	UR3	PushT	NuScenes	Real-world
Obs window size	10	100	3	10	5	1	6
Goal window size (Conditional Task)	10	10	3	10	5	1	-
Predicted act sequence length	1	1	1	10	5	6	1
Autoregressive code pred.	False	False	False	False	False	True	True
β (Eq. 5.4)	0.1	0.6	0.1	0.1	0.1	0.1	0.5
Training Epoch	1000	300	1500	300	2000	1000	600
Learning rate	5.5e-5	5.5e-5	1e-4	5.5e-5	5.5e-5	5.5e-5	3e-4
MinGPT layer num	6	6	4	6	6	6	6
MinGPT head num	6	6	4	6	6	6	6
MinGPT embed dims	120	120	72	120	120	120	120
VQ-VAE latent dims	512	512	256	512	512	512	512
VQ-VAE codebook size	16	10	8	16	16	10	8/10/16
Encoder (Image env)	ResNet18	-	-	-	ResNet18	-	HPR

Table D.6: Hyperparameters for VQ-BeT

D.3.2 VQ-BeT for Driving Dataset

While all the other environments reported in this paper have a fixed observation dimension at one timestep, NuScenes driving dataset, as processed in the GPT-Driver paper [Mao et al. 2023a], could contain the different number of detected objects in each scene. Thus, we make modification to the input types of VQ-BeT to train VQ-BeT with NuScenes driving dataset in response to this change in dimensionality of the obeservation data. The tokens we pass to VQ-BeT are as shown below:

- **Mission Token** indicates the mission that the agent should follow: go forward / turn left / turn right
- **Ego-state Token** contains velocity, angular velocity, acceleration, heading speed, and steering angle.

- **Trajectory History Token** contains ego historical trajectories of last 2 seconds, and ego historical velocities of last 2 seconds.
- **Object Tokens** contains perception and prediction outputs corresponding to current position, predicted future position, and one-hot encoded class indicator of each object. There are total of 15 classes. (‘pushable-pullable’, ‘car’, ‘pedestrian’, ‘bicycle’, ‘truck’, ‘traffic-cone’, ‘motorcycle’, ‘barrier’, ‘bus’, ‘bicycle-rack’, ‘trailer’, ‘construction’, ‘debris’, ‘animal’, ‘emergency’)

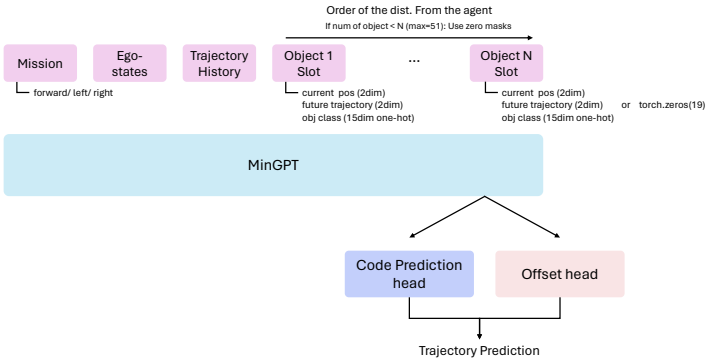


Figure D.5: Overview of VQ-BeT for autonomous driving.

APPENDIX E

Appendix for Robot Utility Models

E.1 Experiment Details

E.1.1 Multimodal Large Language Model Prompts for Success Verification

Here, we present the prompt that we use to verify RUMs success with mLLMs.

Door Opening

As the timesteps progress, does the robotic arm open the door AND is the robot arm grasping the handle in the LAST timestep?

Please respond with only 'Yes' or 'No'.

Drawer Opening

As the timesteps progress, does the robotic arm grasp the drawer handle and open it AND is the drawer open in the last timestep?

Please respond with only 'Yes' or 'No'.

Reorientation

As the timesteps progress, does the robotic arm/gripper reorient the object upright AND is the object upright in the LAST frame?

Please respond with only 'Yes' or 'No'.

Tissue Pick-Up

As the timesteps progress, does the robotic arm/gripper grasp the tissue AND is the gripper grasping the tissue in the LAST timestep?

Please respond with only 'Yes' or 'No'.

Bag Pick-Up

As the timesteps progress, does the robotic arm/gripper grasp the bag AND is the gripper grasping the bag in the LAST timestep?

Please respond with only 'Yes' or 'No'.

E.1.2 Evaluation Schedule

In Figure E.1, we show the starting position of the robot for our 10-run evaluations to understand the positional generalization capabilities of Robot Utility Models.

E.1.3 Failure Modes

As we mention in the main paper, with mLLM guided retries, our failures tend to be more peculiar than simply "robot failed to complete task". In this section, we try to shine some light on what kind of failures we experience in our system.

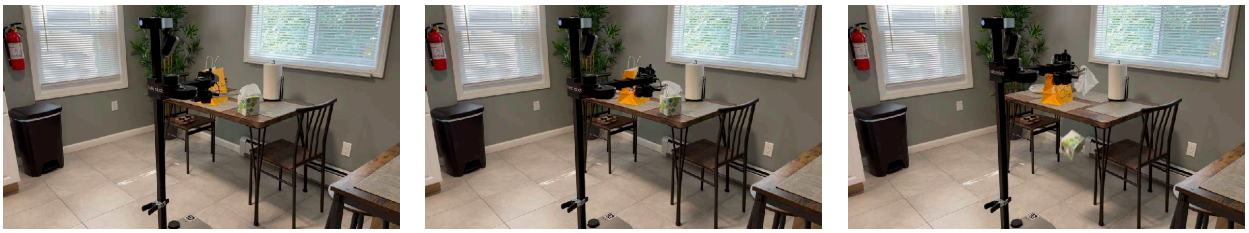
Figure E.1: 10-run evaluation schedule used to evaluate Robot Utility Models, with robot starting positions denoted by the pale blue dots in the image. We assume that the robot is at the task space facing the object, but it can be at different offsets with respect to the target object. On our object centric tasks (reorientation, bag and tissue pickup) we also randomize the position of the object itself.

- **Reorientation:** Primary failure modes for this task are when retry becomes impossible because of environmental issues, such as the target bottle rolling away on the table, being dropped off the surface (an example of which is shown on the Figure E.2), pushing it too far into the table (to a position too far for our robot arm), or being rotated sideways by the gripper before grasping. In out-of-distribution surfaces, it can be hard to estimate how large the surface is visually and thus placing the object after reorientation may miss the surface or the robot may run into the surface.

- **Drawer opening:** Beyond the most direct failure mode of missing the drawer handle, we experienced some failure modes where the model does not know when to stop pulling on cloth drawers and thus pulls out the entire drawer. Without force feedback, it can be hard to tell visually when the drawer starts sagging. Force feedback on the fingertips would help the robot correct for it.

- **Door opening:** Here, the primary failure mode we experience are on unusual doors, such as

Reorientation failure: dropped bottle off the table, retry impossible



Tissue pick up failure: picked up tissue, pulled box off the table

Tissue pick up failure: picked up tissue AND the box

Figure E.2: Examples of some failures in real world rollouts. Since RUMs retries on failure with mLLM feedback, the failure modes tend to be peculiar, some examples of which are shown here.

the trash cabinet door with a hole in it. There, GPT sometimes classifies the door as "open" even when it is closed. In some rare cases, when door handles are close together, the robot may grasp around both handles and fail to reset as it gets stuck when retracting.

- **Tissue pick up:** The tissue box itself being light and easy to move means that sometimes the box moves with the tissue as its being picked up. As a result, the box may get picked up with the tissue, or get pushed off from its table by the robot (Figure E.2.)

- **Bag pick up:** The case of bag picking up is interesting because it has one of the highest success rates from the raw RUM policy but also sees the smallest improvement (4%) from GPT feedback. This failure from mLLM feedback happens usually because from the robot wrist or head camera,

it can be hard to tell whether the bag has been picked up. As a result, GPT tends to have a high number of false positives for this task. Having a better third-person view of the workspace should help address this issue.

E.1.4 Detailed Results from Experiments with Self-critique and Retrying

Task	Environment/Object	Success ·/10
Door Opening	Kitchen Trash Door	7
	Kitchen Cabinet Door	10
	Brown Cabinet Door	10
	Metal Cabinet Door	10
	White File Cabinet Door	10
Drawer Opening	Kitchen Drawer	10
	Cloth Drawer	9
	White File Cabinet Drawer	10
	Small File Cabinet Drawer	10
	Dresser Drawer	8
Bag Pick Up	Hollister Bag	9
	American Eagle Bag	10
	Qdoba Bag	8
	Journey's Bag	9
	Yellow Bag	6
Tissue Pick Up	White Tall Box	10
	White Short Box	10
	Black Square Box	9
	Red Square Box	10
	Kleenex Box	7
Object Reorientation	Pink Bottle	9
	White Board Cleaner	8
	Spices Container	8
	Coke Can	8
	Compressed Air	10

Table E.1: Detailed success statistics of RUMs on our evaluation environments.

E.2 Hardware and Physical Setup

E.2.1 Bill of Materials

Here, we present the bill of materials for our hardware components, assuming that the interested researcher or user owns an iPhone Pro already. The total cost comes out to be slightly below $25 for the entire setup.

Item	Price	Unit Price	Qty
Reacher Grabber Tool	26.99	13.50	1
Brass Tapered Heat-Set Inserts	21.82	0.22	3
Thread-Forming Screws	7.75	0.31	3
Button Head Screw - M4 x 0.70 - 8mm	12.91	0.13	1
Button Head Screw - M4 x 0.70 - 5mm	8.64	0.09	2
Button Head Screw - M4 x 0.70 - 35mm	16.77	0.34	2
Nylon-Insert Locknut	5.57	0.06	2
Dowel Pin	16.09	0.32	3
Nylon Unthreaded Spacer	18.41	0.18	2
Kevlar Cord	20.99	20.99	1/100
Heat Shrink Tubing	10.79	10.79	1/30
Black 3D Printer Filament	25.99	25.99	3/20
Total		**21.99**	

Table E.2: Stick-v2 Main Body

Item	Price	Unit Price	Qty
Socket Head Screw - M3 x 0.5mm - 8mm	12.52	0.13	2
Steel Hex Nut - M3 x 0.5mm	2.62	0.03	2
M3 Steel Washer	2.19	0.02	2
Red 3D Printer Filament	25.99	25.99	3/1000
Oomoo 25 Silicone Rubber	33.99	33.99	1/200
Total		**0.61**	

Table E.3: Gripper Tips

Item	Price	Unit Price	Qty
Socket Head Screw - M5 x 0.8mm - 20mm	17.10	0.17	1
Socket Head Screw - M5 x 0.8mm - 50mm	4.26	0.85	1
Steel Hex Nut - M5 x 0.8mm	5.24	0.05	2
Button Head Screw - M4 x 0.70 - 8mm	12.91	0.13	1
Black 3D Printer Filament	25.99	25.99	3/20
Total		**2.03**	

Table E.4: Phone Holder

E.2.2 Deploying on Stretch's Default D405 Camera

Deploying our Robot Utility Models on the standard Hello Robot Stretch SE3 requires normalizing the image coming out of the default Intel Realsense D405 wrist camera. We created an affine transformation that maps the D405 image to the same pixel coordinates as the iPhone camera.

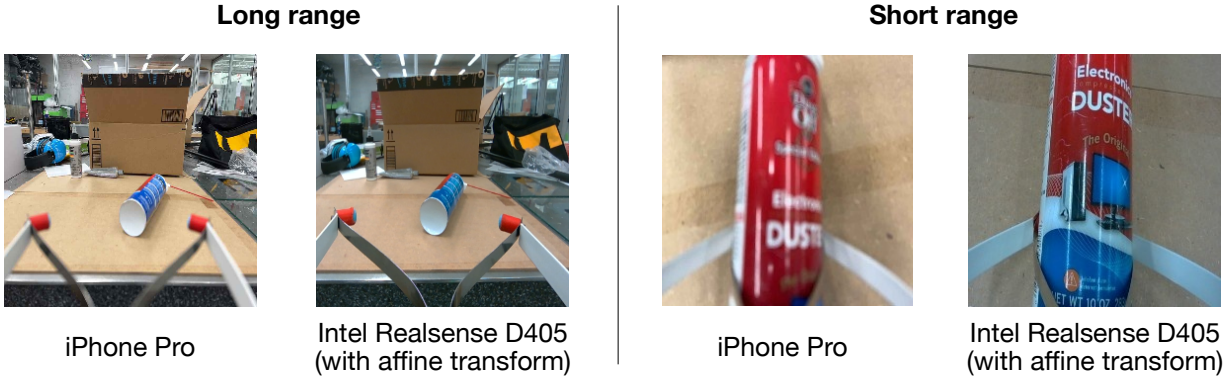


Figure E.3: We can see the corresponding D405 camera image alongside the iPhone Pro image. While in the long range, the images look similar, in the short range iPhone images are out of focus because of the different focal lengths of the cameras.

As we can see from Figure E.3, applying the affine transform to the D405 camera maps it to pretty similar viewpoint as the wrist mounted iPhone. While we can run RUMs directly with this camera transform, we see a performance drop which we hypothesize happens because of the especially apparent difference in close-range. This difference is caused by the different focal lengths of the two cameras, and may be solved in the future with image augmentations.

E.2.3 Evaluation Environments

Figure E.4: Picture of evaluation environments for the tasks Reorientation, Drawer opening, and Door opening.

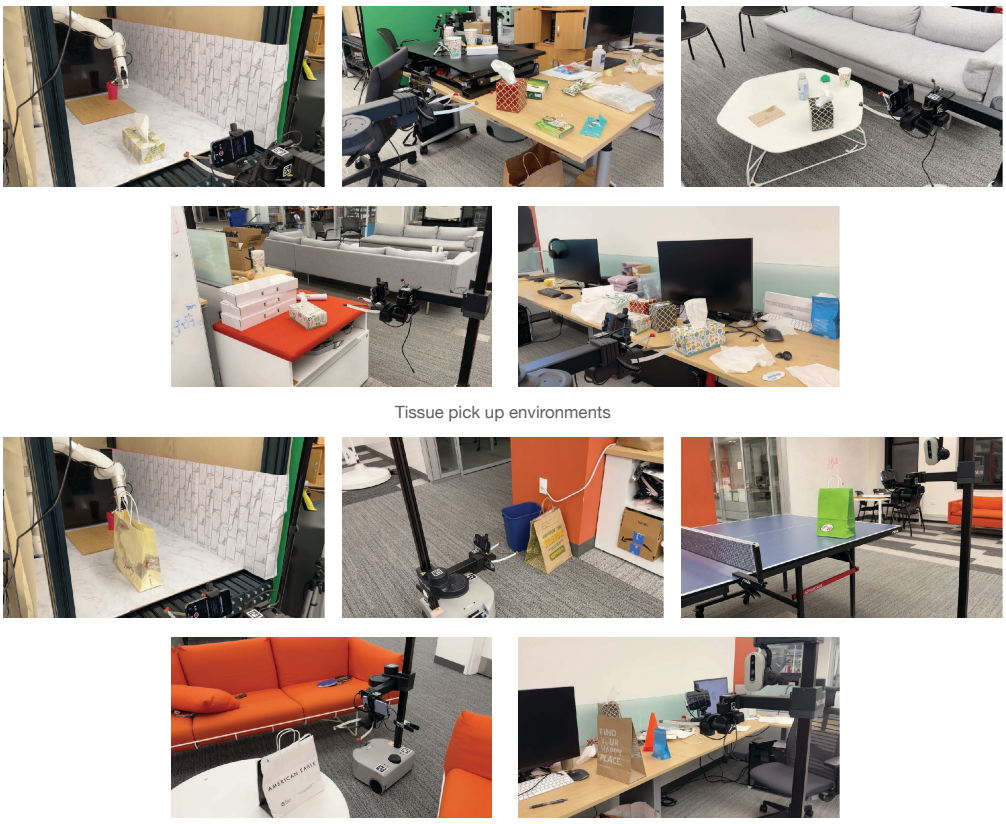


Figure E.5: Pictures of the evaluation environments for the task Tissue pick up and Bag pick up.

APPENDIX F

Appendix for CLIP-Fields

F.1 Training details

We release our open source code at the Github repo https://github.com/clip-fields/clip-fields.github.io with full details about how to train a new CLIP-Fields on any environment. The code is also shared in the attached supplementary information zip file. While the published code should be sufficient to reproduce our work and experiments, we are describing the most important training details and hyperparameters here for reproducibility purposes.

Table F.1: Optimization hyperparameters

Parameter	Value
Optimizer	Adam
Learning rate	10^{-4}
Weight decay	3×10^{-3}
β	$(0.9, 0.999)$
Learning rate schedule	None
Epochs	100
Per epoch iters	3×10^{6}
Batch size	12, 544
α (Sec. 9.4.4, when applicable)	100.0

Table F.2: Architecture and Instant-NGP hyperparameters

Parameter	Value
Intermediate representation dimension	144
NGP grid levels	18
NGP per-level scale	2
NGP level dimension	8
NGP $\log_2$ hash map size	20
MLP number of hidden layers	1
MLP hidden layer size	600

Table F.3: External model configurations

Task	Model	Instance
Object detector	Detic	CLIP + SwinB
Vision-language model	CLIP	ViT-B/32
Language model	Sentence-BERT	all-mpnet-base-v2

F.2 Real world experiment logs

In this section, we reproduce the exact real-world qualitative observations that we made by running our robot on the Kitchen scenario. We present this for the readers to get a full picture of what the robot queries looked like, and how the CLIP-Fields responded to each of the queries.

1. Literal queries:
   (a) Stack of plates: success, found the dishwashing rack with plates in it.
   (b) Microwave: success, found the microwave oven in the lab kitchen.
   (c) The fridghe (misspelling intentional): success, found the large standing fridge in the corner.
   (d) Coffee machine (ambiguous query): success, found the silver coffee maker.
   (e) Sink: success, found the sink.
   (f) Toaster oven: failure, found the microwave oven instead of the toaster oven.

2. Visual queries:

    (a) White ceramic bowl: success, found the bowl by the dishwashing rack.

    (b) Red plastic bowl: success, found the red bowl above the trash cabinets.

    (c) Red fruit bowl: failure, found the white bowl by the dishwashing rack.

    (d) Espresso machine: success, found the nespresso machine by the coffee machine.

    (e) Blue gargabe bin: success, found one of the two blue recycling bins in the kitchen.

    (f) Potted plant in a black pot: success, ambiguous, found the potted plants in a shelf. Isolating the black flower pot was ambiguous since the robot doesn't get too close to scene objects.

    (g) Purple poster: success, found the poster above the sink.

3. Semantic queries:

    (a) Wash my dishes: success, finds the dishwasher as intended.

    (b) Wash my hand: failure, finds the dishwasher instead of the sink.

    (c) Throw my trash: success, finds the recycling bins (although not entirely climate friendly behavior.)

    (d) Put away my leftovers: failure, pointed the camera at the trash cabinet instead of the fridge or the cabinets. Potentially because the trash cabinets got identified as "cabinets" by our detectors.

    (e) Fill out my water bottle: success, finds the glass bottles at the corner of the kitchen. While the original intention was to find the water cooler, the response is reasonable.

    (f) Make some coffee: success, found the coffee maker and grinders.

    (g) Warm up my lunch: success, found the microwave oven.

APPENDIX G

Appendix for OK Robot

G.1 Description of alternate system components

In this section, we provide more details about the alternate system components that we evaluated in Section 10.3.2.

G.1.1 Alternate semantic navigation strategies

We evaluate the following semantic memory modules:

- **VoxelMap [Yenamandra et al. 2023b]:** VoxelMap converts every detected object to a semantic vector and stores such info into an associated voxel. Occupied voxels serve as an obstacle map.
- **CLIP-Fields [Shafiullah et al. 2023a]:** CLIP-Fields converts a sequence of posed RGB-D images to a semantic vector field by using open-label object detectors and semantic language embedding models. The result associates each point in the space with two semantic vectors, one generated via a VLM [Radford et al. 2021], and another generated via a language model [Reimers and Gurevych 2019], which is then embedded into a neural field [Mildenhall et al. 2020].

- **USA-Net [Bolte et al. 2023]:** USA-Net generates multi-scale CLIP features and embeds them in a neural field that also doubles as a signed distance field. As a result, a single model can support both object retrieval and navigation.

We compare them in the same three environments with a fixed set of queries, the results of which are shown in Figure 10.5.

G.1.2 Alternate grasping strategies

Similarly, we compare multiple grasping strategies to find out the best grasping strategy for our method.

- **AnyGrasp [Fang et al. 2023c]:** AnyGrasp is a single view RGB-D based grasping model. It is trained on the GraspNet dataset which contains 1B grasp labels.
- **Open Graspness [Fang et al. 2023c]:** Since the AnyGrasp model is free but not open source, we use an open licensed baseline trained on the same dataset.
- **Contact-GraspNet [Sundermeyer et al. 2021]:** We use Contact-GraspNet as a prior work baseline, which is trained on the Acronym [Eppner et al. 2021] dataset. One limitation of Contact-GraspNet is that it was trained on a fixed camera view for a tabletop setting. As a result, in our application with a moving camera and arbitrary locations, it failed to give us meaningful grasps.
- **Top-down grasp [Yenamandra et al. 2023b]:** As a heuristic based baseline, we compare with the top-down heuristic grasp provided in the HomeRobot project.

G.2 Scannet200 text queries

To detect objects in a given home environment using OWL-ViT, we use the Scannet200 labels. The full label set is here: ['shower head', 'spray', 'inhaler', 'guitar case', 'plunger', 'range hood', 'toilet paper dispenser', 'adapter', 'soy sauce', 'pipe', 'bottle', 'door', 'scale', 'paper towel', 'paper towel roll', 'stove', 'mailbox', 'scissors', 'tape', 'bathroom stall', 'chopsticks', 'case of water bottles', 'hand sanitizer', 'laptop', 'alcohol disinfection', 'keyboard', 'coffee maker', 'light', 'toaster', 'stuffed animal', 'divider', 'clothes dryer', 'toilet seat cover dispenser', 'file cabinet', 'curtain', 'ironing board', 'fire extinguisher', 'fruit', 'object', 'blinds', 'container', 'bag', 'oven', 'body wash', 'bucket', 'cd case', 'tv', 'tray', 'bowl', 'cabinet', 'speaker', 'crate', 'projector', 'book', 'school bag', 'laundry detergent', 'mattress', 'bathtub', 'clothes', 'candle', 'basket', 'glass', 'face wash', 'notebook', 'purse', 'shower', 'power outlet', 'trash bin', 'paper bag', 'water dispenser', 'package', 'bulletin board', 'printer', 'windowsill', 'disinfecting wipes', 'bookshelf', 'recycling bin', 'headphones', 'dresser', 'mouse', 'shower gel', 'dustpan', 'cup', 'storage organizer', 'vacuum cleaner', 'fireplace', 'dish rack', 'coffee kettle', 'fire alarm', 'plants', 'rag', 'can', 'piano', 'bathroom cabinet', 'shelf', 'cushion', 'monitor', 'fan', 'tube', 'box', 'blackboard', 'ball', 'bicycle', 'guitar', 'trash can', 'hand sanitizers', 'paper towel dispenser', 'whiteboard', 'bin', 'potted plant', 'tennis', 'soap dish', 'structure', 'calendar', 'dumbbell', 'fish oil', 'paper cutter', 'ottoman', 'stool', 'hand wash', 'lamp', 'toaster oven', 'music stand', 'water bottle', 'clock', 'charger', 'picture', 'bascketball', 'sink', 'microwave', 'screwdriver', 'kitchen counter', 'rack', 'apple', 'washing machine', 'suitcase', 'ladder', 'ping pong ball', 'window', 'dishwasher', 'storage

container', 'toilet paper holder', 'coat rack', 'soap dispenser', 'refrigerator', 'banana', 'counter', 'toilet paper', 'mug', 'marker pen', 'hat', 'aerosol', 'luggage', 'poster', 'bed', 'cart', 'light switch', 'backpack', 'power strip', 'baseball', 'mustard', 'bathroom vanity', 'water pitcher', 'closet', 'couch', 'beverage', 'toy', 'salt', 'plant', 'pillow', 'broom', 'pepper', 'muffins', 'multivitamin', 'towel', 'storage bin', 'nightstand', 'radiator', 'telephone', 'pillar', 'tissue box', 'vent', 'hair dryer', 'ledge', 'mirror', 'sign', 'plate', 'tripod', 'chair', 'kitchen cabinet', 'column', 'water cooler', 'plastic bag', 'umbrella', 'doorframe', 'paper', 'laundry hamper', 'food', 'jacket', 'closet door', 'computer tower', 'stairs', 'keyboard piano', 'person', 'table', 'machine', 'projector screen', 'shoe'].

G.3 Sample objects from our trials

During our experiments, we tried to sample objects that can plausibly be manipulated by the Hello Robot: Stretch gripper from the home environments. As a result, OK-Robot encountered a large variety of objects with different shapes and visual features. A subsample of such objects are presented in the Figures G.1, G.2.

G.4 Sample home environments from our trials

We show snapshots from a subset of home environments where we evaluated our method in Figures G.3. Additionally, in Figure G.4 we show the two home environments in Pittsburgh, PA, and Fremont, CA, where we reproduced the OK-Robot system.

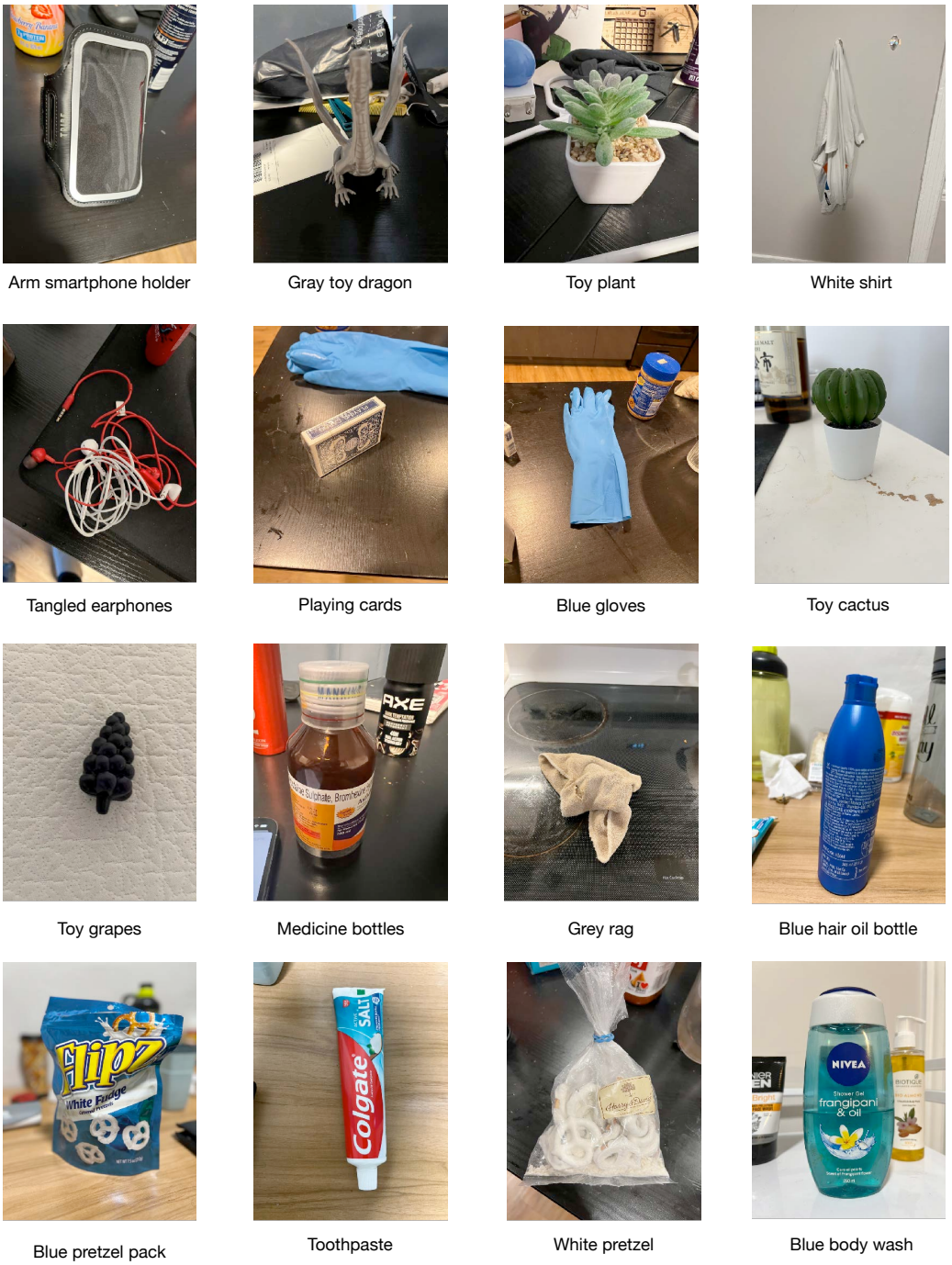


Figure G.1: Sample objects on our home experiments, sampled from each home environment, which OK-Robot was able to pick and drop successfully.

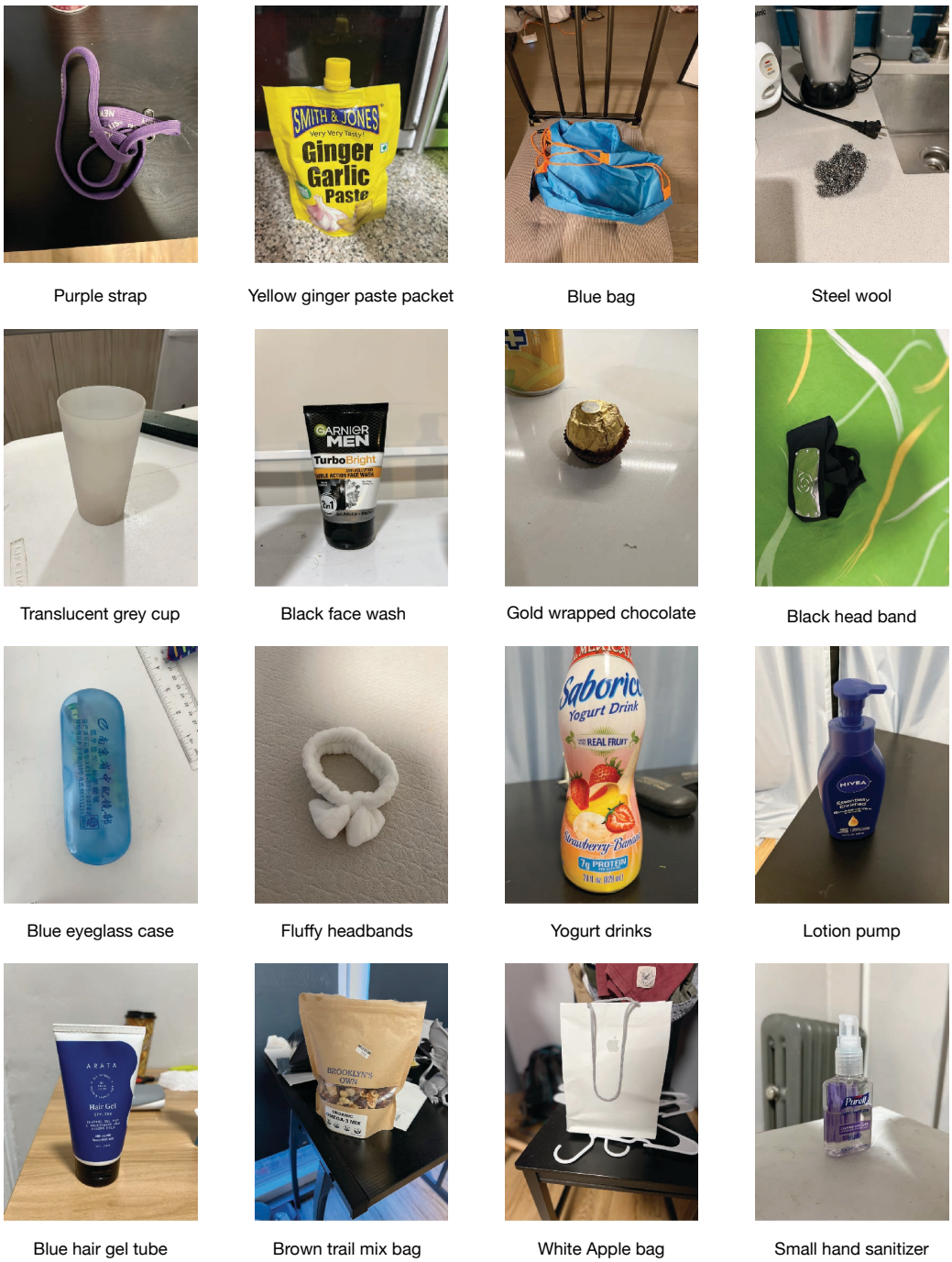


Figure G.2: Sample objects on our home experiments, sampled from each home environment, which `OK-Robot` **failed** to pick up successfully.

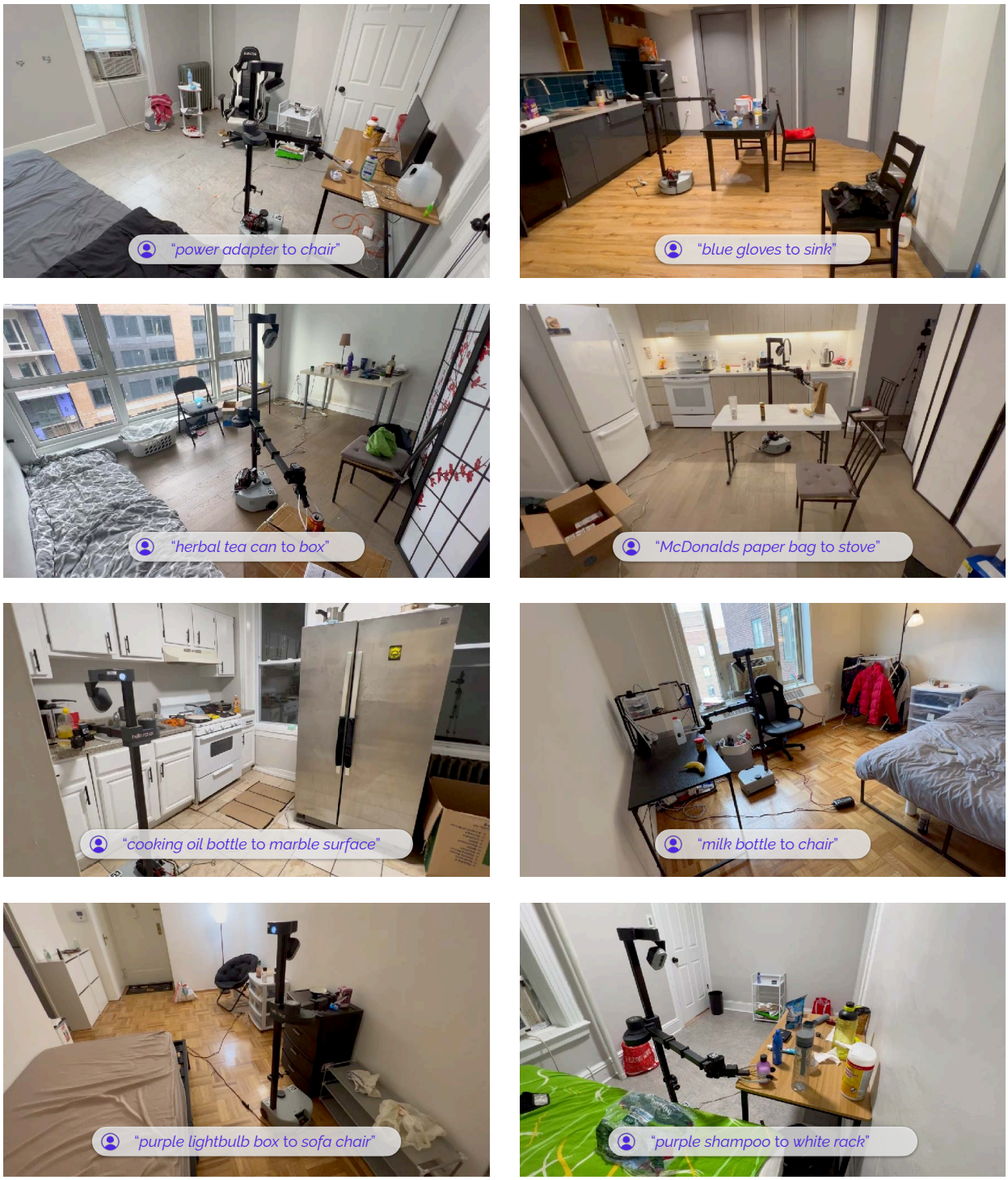
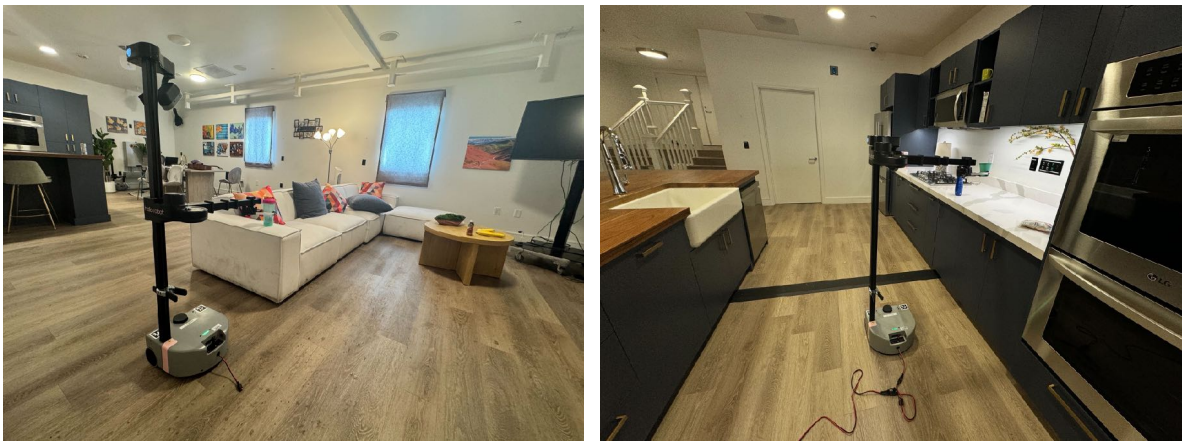


Figure G.3: Eight out of the ten New York home environments in which we evaluated OK-Robot. In each figure caption, we show the queries that the system is being evaluated on.

Reproducibility experiments in Pittsburgh, PA



Reproducibility experiments Fremont, CA

Figure G.4: Home environments outside of New York where we successfully reproduced OK-Robot. We ensured that OK-Robot can function in these homes by trying pick-and-drop on a number of objects in the homes.

G.5 List of home experiments

A full list of experiments in homes can be found in Table G.1.

Table G.1: A list of all tasks in the home enviroments, along with their categories and success rates out of 10 trials.

Pick object	Place location	Result
Home 1		
Cleanup level: none		
silver cup	white table	Success
blue eye glass case	chair	Success
printed paper cup, coffee cup [white table]	____	Manipulation failure
small red and white medication	Chair	Success
power adapter	Grey Bed	Success
wrapped paper	____	Navigation failure
blue body wash	study table	Success
blue air spray	white table	Success
black face wash	____	Manipulation failure
yellow face wash	chair	Success
body spray	____	Navigation failure
small hand sanitizer	____	Manipulation failure
blue inhaler device(window)	white table	Success
inhaler box(window)	dust bin	Success
multivitamin container	____	Navigation failure
red towel	white cloth bin (air conditioner)	Success
white shirt	white cloth bin (air conditioner)	Success
Cleanup level: low		
silver cup	white table	Success
blue eye glass case	____	Navigation failure

Continued on the next page

Pick object	Place location	Result
printed paper cup, coffee cup [white table]	dust bin	Success
small red and white medication	Chair	Success
power adapter	____	Navigation failure
blue body wash	white table	Success
blue air spray	white table	Success
yellow face wash	white table	Success
small hand sanitizer	study table	Success
blue inhaler device(window)	____	Manipulation failure
inhaler box(window)	dust bin	Success
red towel	white cloth bin(air conditioner)	Success
white shirt	white cloth bin(air conditioner)	Success
Cleanup level: high		
silver cup	white table	Success
printed paper cup, coffee cup [white table]	dust bin	Success
blue body wash	white table	Success
blue air spray	white table	Success
yellow face wash	____	Manipulation failure
small hand sanitizer	____	Manipulation failure
inhaler box(window)	dust bin	Success
white shirt	white cloth bin(air conditioner)	Success
Home 2		
Cleanup level: None		
fanta can	dust bin	Success
tennis ball	small red shopping bag	Success
black head band [bed]	____	Manipulation failure
purple shampoo bottle	white rack	Success
toothpaste	small red shopping bag	Success

Continued on the next page

Pick object	Place location	Result
orange packaging	dust bin	Success
green hair cream jar [white rack]	____	Navigation failure
green detergent pack [white rack]	white table	Success
blue moisturizer [white rack]	____	Navigation failure
green plastic cover	____	Navigation failure
storage container	____	Manipulation failure
blue hair oil bottle	white rack	Success
blue pretzels pack	white rack	Success
blue hair gel tube	____	Manipulation failure
red bottle [white rack]	brown desk	Success
blue bottle [air conditioner]	white cloth bin(air conditioner)	Success
wallet	____	Manipulation failure
Cleanup level: low		
fanta can	black trash can	Success
tennis ball	red target bag	Success
black head band [bed]	red target bag	Success
purple shampoo bottle	red target bag	Success
toothpaste	red target bag	Success
orange packaging	black trash can	Success
green detergent pack [white rack]	____	Manipulation failure
blue moisturizer [white rack]	____	Navigation failure
blue hair oil bottle	white rack	Success
blue pretzels pack	white rack	Success
wallet	____	Manipulation failure
Cleanup level: high		
fanta can	black trash can	Success
purple shampoo bottle	small red shopping bag	Success

Continued on the next page

Pick object	Place location	Result
orange packaging	black trash can	Success
blue moisturizer [white rack]	white rack	Success
blue hair oil bottle	____	Manipulation failure
blue hair gel tube	dust bin	Success
red bottle [white rack]	target bag	Placing failure
blue bottle [air conditioner]	white cloth bin(air conditioner)	Success

Home 3		
Cleanup level: none		
apple	white plate	Success
ice cream	white and green bag	Success
green lime juice bottle	red basket	Success
yellow packet	____	Manipulation failure
red packet	____	Manipulation failure
orange can	card board box	Success
cooking oil bottle	____	Manipulation failure
pasta sauce	____	Manipulation failure
orange box [stove]	____	Manipulation failure
green bowl	sink	Success
washing gloves	green bag [card board box]	Success
small oregano bottle	red basket	Success
yellow noodles packet [stove]	red basket	Success
blue dish wash bottle	card board box	Success
scrubber	____	Navigation failure
dressing salad bottle	____	Navigation failure
Cleanup level: low		
apple	white plate	Success
ice cream	red basket	Success

Continued on the next page

Pick object	Place location	Result
green lime juice bottle	red basket	Success
yellow packet	green bag	Success
red packet	____	Manipulation failure
orange can	card board box	Success
cooking oil bottle	marble surface [red basket]	Success
green bowl	____	Manipulation failure
washing gloves	sink	Success
small oregano bottle	red basket	Success
yellow noodles packet [stove]	____	Manipulation failure
blue dish wash bottle	card board box	Success
Cleanup level: high		
apple	white plate	Success
ice cream	red basket	Success
green lime juice bottle	red basket	Success
orange can	card board box	Success
cooking oil bottle	____	Manipulation failure
washing gloves	sink	Success
small oregano bottle	red basket	Success
yellow noodles packet [stove]	red basket	Success
blue dish wash bottle	card board box	Success
Home 4		
Cleanup level: none		
pepsi	black chair	Success
birdie	cloth bin	Success
black hat	____	Navigation failure
owl like wood carving	bed	Success
red inhaler	____	Manipulation failure

Continued on the next page

Pick object	Place location	Result
black sesame seeds	____	Manipulation failure
banana	____	Manipulation failure
loose-leaf herbal tea jar	black chair	Success
red pencil sharpener	____	Navigation failure
fast-food French fries container	blue shopping bag [metal drying rack]	Placing failure
milk	plastic storage drawer unit	Success
socks[bed]	____	Navigation failure
purple gloves	____	Manipulation failure
target bag	cloth bin	Success
muffin	grey bed	Success
tissue paper	table	Success
grey eyeglass box	____	Manipulation failure
Cleanup level: low		
pepsi	basket	Success
birdie	white drawer	Success
owl like wood carving	____	Navigation failure
red inhaler	plastic storage drawer unit	Success
black sesame seeds	bed	Success
loose-leaf herbal tea jar	table	Success
fast-food French fries container	chair	Success
milk	chair	Success
purple gloves	basket	Success
target bag	basket	Placing failure
muffin	table	Success
tissue paper	____	Manipulation failure
grey eyeglass box	____	Navigation failure
Cleanup level: high		

Continued on the next page

Pick object	Place location	Result
pepsi	basket	Success
birdie	bed	Success
red inhaler	plastic storage drawer unit	Success
black sesame seeds	desk	Success
banana	____	Manipulation failure
loose-leaf herbal tea jar	____	Manipulation failure
milk	chair	Success
purple gloves	basket	Success
target bag	basket	Success
muffin	bed	Success

Home 5		
Cleanup level: none		
tiger balm topical ointment	____	Navigation failure
pink shampoo	trader joes shapping bag	Success
aveeno sunscreen protective lotion	trader joes shapping bag	Success
small yellow nozzle spray	____	Manipulation failure
black hair care spray	____	Manipulation failure
green hand sanitizer	____	Manipulation failure
white hand sanitizer	____	Navigation failure
white bowl [ketchup]	black sofa chair	Success
blue bowl	____	Manipulation failure
blue sponge	trader joes shapping bag	Success
ketchup	____	Manipulation failure
white salt	____	Manipulation failure
black pepper	black drawer	Success
blue bottle	____	Navigation failure
purple light bulb box	trader joes shopping bag	Success
white plastic bag	bed	Success

Continued on the next page

Pick object	Place location	Result
rag	white rack	Success
Cleanup level: low		
pink shampoo	____	Navigation failure
aveeno sunscreen protective lotion	_____	Manipulation failure
small yellow nozzle spray	_____	Manipulation failure
white bowl [ketchup]	black sofa chair	Success
blue sponge	bed	Success
ketchup	trader joes shopping bag	Success
white salt	trader joes shopping bag	Success
black pepper	____	Navigation failure
blue bottle	black sofa chair	Success
purple light bulb box	_____	Manipulation failure
rag	white rack	Success
Cleanup level: high		
pink shampoo	trader joes shopping bag	Success
green hand sanitizer	black sofa chair	Success
white bowl [ketchup]	_____	Manipulation failure
blue sponge	bed	Success
ketchup	black drawer	Success
white salt	white drawer	Success
purple light bulb box	trader joes shopping bag	Success
rag	black sofa chair	Success
Home 6		
Cleanup level: none		
translucent grey cup	____	Manipulation failure
green mouth spray box	stove	Success

Continued on the next page

Pick object	Place location	Result
green eyeglass container	chair	Success
blue bag	____	Manipulation failure
black burn ointment box	_____	Navigation failure
white vitamin bottle	____	Navigation failure
McDonald's paper bag	stove	Success
purple medicine packaging	chair	Success
grey rag	sink	Success
sparkling water can [sink]	countertop	Success
gold wrapped chocolate	____	Manipulation failure
lemon tea carton	table	Success
metallic golden beverage can	table	Success
red bottle	table	Success
tea milk bottle	____	Navigation failure
nyu water bottle [sink]	table	Success
white hand wash	_____	Navigation failure
Cleanup level: low		
translucent grey cup	____	Navigation failure
green mouth spray box	____	Manipulation failure
blue bag	brown box	Success
black burn ointment box	brown box	Success
McDonald's paper bag	____	Navigation failure
grey rag	sink	Success
sparkling water can [sink]	chair	Success
lemon tea carton	stove	Success
metallic golden beverage can	____	Navigation failure
red bottle	brown box	Success
nyu water bottle [sink]	table	Success
white hand wash	sink	Success

Continued on the next page

Pick object	Place location	Result
Cleanup level: high		
blue bag	brown box	Success
black burn ointment box	_____	Manipulation failure
grey rag	sink	Success
sparkling water can [sink]	chair	Success
lemon tea carton	table	Success
metallic golden beverage can	stove	Success
red bottle	_____	Navigation failure
nyu water bottle [sink]	_____	Manipulation failure
white hand wash	_____	Manipulation failure
Home 7		
Cleanup level: none		
blue plastic bag roll	_____	Navigation failure
green bag	basket[window]	Success
toy cactus	desk	Success
toy van	chair	Success
brown medical bandage	chair	Success
power adapter	_____	Navigation failure
red herbal tea	brown cardboard box	Success
apple juice box	brown cardboard box	Success
paper towel	blue cardboard box	Success
toy bear	bed blanket	Success
yellow ball	bed blanket	Success
black pants	basket[window]	Success
purple water bottle	desk	Success
blue eyeglass case	_____	Manipulation failure
brown toy monkey	_____	Navigation failure

Continued on the next page

Pick object	Place location	Result
blue hardware box [table]	blue cardboard box	Success
green zandu balm container	blue cardboard box	Success
Cleanup level: low		
green bag	basket	Success
toy cactus	basket	Success
toy van	chair	Success
brown medical bandage	_____	Manipulation failure
red herbal tea	brown box	Success
apple juice box	brown box	Success
paper towel	basket	Success
toy bear	desk	Success
purple water bottle	desk	Success
blue eyeglass case	_____	Manipulation failure
green zandu balm container	blue cardboard box	Success
Cleanup level: high		
green bag	stool [window]	Success
toy cactus	table	Success
toy van	white basket	Success
red herbal tea	brown cardboard box	Success
apple juice box	brown cardboard box	Success
paper towel	blue cardboard box	Success
toy bear	white basket	Success
yellow ball	bed	Success
purple water bottle	black tote bag	Success
green zandu balm container	blue cardboard box	Success
Home 8		

Continued on the next page

Pick object	Place location	Result
Cleanup level: none		
cyan air spray	brown shelf [sink]	Success
blue gloves	kitchen sink	Success
blue peanut butter	black stove	Success
nutella	table	Success
green bag	brown shelf [sink]	Success
green bandage box	trash can	Success
green detergent	kitchen sink	Success
black ‘red pepper sauce’	____	Manipulation failure
red bag	chair	Success
black bag	chair	Success
red spray [brown shelf]	kitchen countertop	Success
steel wool	_____	Manipulation failure
white aerosol	trash can	Success
white pretzel	black stove	Success
purple crisp	kitchen countertop	Success
plastic bowl	_____	Manipulation failure
playing card	microwave	Success
Cleanup level: low		
cyan air apray	chair	Success
blue gloves	sink	Success
blue peanut butter	____	Navigation failure
green bag	brown shelf	Success
green bandage box	brown shopping bag	Success
green detergent	microwave	Success
red bag	____	Manipulation failure
black bag	chair	Success
white aerosol	trash can	Success

Continued on the next page

Pick object	Place location	Result
white pretzel	black stove	Success
purple crisp	kitchen countertop	Success
plastic bowl	_____	Manipulation failure
playing card	microwave	Success
Cleanup level: high		
cyan air apray	brown shelf [sink]	Success
blue gloves	stove	Success
blue peanut butter	black stove	Success
green bag	brown shelf [sink]	Success
green bandage box	microwave	Success
green detergent	____	Manipulation failure
black bag	chair	Success
white aerosol	table	Success
purple crisp	chair	Success
playing card	microwave	Success
Home 9		
Cleanup level: none		
toy grapes	black laundry bag	Success
purple strap	_____	Manipulation failure
red foggy body spray	_____	Manipulation failure
arm smartphone holder	bed	Success
medicine bottle	_____	Manipulation failure
yogurt beverage	_____	Navigation failure
blue shaving cream can	____	Navigation failure
blue cup	table	Success
purple tape	_____	Manipulation failure
black shoe brush	_____	Navigation failure

Continued on the next page

Pick object	Place location	Result
fluffy headband	_____	Manipulation failure
black water bottle	brown shopping bag	Placing failure
yellow eyeglass case	black chair	Success
paper cup	_____	Manipulation failure
lotion pump	_____	Manipulation failure
nasal spray	_____	Manipulation failure
plastic bag	trash basket	Success
Cleanup level: low		
toy grapes	_____	Manipulation failure
red foggy body spray	brown paper bag	Success
arm smartphone holder	brown paper bag	Success
yogurt beverage	desk	Success
blue shaving cream can	black bag	Success
blue cup	black chair	Success
black shoe brush	_____	Manipulation failure
fluffy headband	_____	Navigation failure
black water bottle	folded chair	Success
nasal spray	_____	Navigation failure
plastic bag	trash basket	Success
Cleanup level: high		
red foggy body spray	brown paper bag	Success
arm smartphone holder	_____	Manipulation failure
yogurt beverage	desk	Success
blue shaving cream can	black bag	Success
blue cup	black chair	Success
black water bottle	white bed	Success
nasal spray	folded chair	Success

Continued on the next page

Pick object	Place location	Result
plastic bag	trash basket	Success
Home 10		
Cleanup level: none		
grey toy dragon	bed	Success
purple body spray	____	Manipulation failure
hand sanitizer	shelf	Success
toy plant	bed [shelf]	Success
brown trail mix bag	____	Manipulation failure
hanging blue shirt	cloth bin	Success
white apple bag	____	Manipulation failure
white and pink powder bottle	table	Success
cough syrup bottle	shelf	Success
tangled ear phones	office chair	Success
red deodrant stick[table]	chair	Success
black body spray	chair	Success
hair treatment medicine bottle	____	Manipulation failure
green tea package	chair	Success
portable speaker [green tea package]	office chair	Success
wooden workout gripper	_____	Navigation failure
brown box	_____	Navigation failure
blue bulb adapter	office chair	Success
game controller	office chair	Success
Cleanup level: low		
grey toy dragon	orange bag	Success
purple body spray	table	Success
hand sanitizer	_____	Navigation failure
toy plant	bed	Success

Continued on the next page

Pick object	Place location	Result
brown trail mix bag	____	Manipulation failure
white and pink powder bottle	black chair [bed]	Success
cough syrup bottle	shelf [bed]	Success
red deodrant stick[table]	bed [rack]	Success
black body spray	rack [bed]	Placing failure
green tea package	orange bag	Success
brown box	black chair [bed]	Success
blue bulb adapter	_____	Manipulation failure
Cleanup level: high		
purple body spray	orange bag	Success
toy plant	bed	Success
white and pink powder bottle	_____	Navigation failure
cough syrup bottle	shelf [bed]	Success
red deodrant stick[table]	_____	Navigation failure
black body spray	black chair	Success
green tea package	table	Success
blue bulb adapter	shelf	Success

Bibliography

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Adorjan, M. (2016). *OpenSfM: A Collaborative Structure-From-Motion System*. PhD thesis, Wien.

Agarwal, A., Kumar, A., Malik, J., and Pathak, D. (2023). Legged locomotion in challenging terrains using egocentric vision. In Liu, K., Kulic, D., and Ichnowski, J., editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 403–415. PMLR.

Aha, D. W. and Salzberg, S. L. (1994). Learning to catch: Applying nearest neighbor algorithms to dynamic control tasks. In Cheeseman, P. and Oldford, R. W., editors, *Selecting Models from Data*, pages 321–328, New York, NY. Springer New York.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. (2019). Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Binkowski, M., Barreira,

R., Vinyals, O., Zisserman, A., and Simonyan, K. (2022). Flamingo: a visual language model for few-shot learning.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *NIPS*, 30:5048–5058.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Arunachalam, S. P., Güzey, I., Chintala, S., and Pinto, L. (2023a). Holo-dex: Teaching dexterity with immersive mixed reality. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5962–5969. IEEE.

Arunachalam, S. P., Silwal, S., Evans, B., and Pinto, L. (2023b). Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation. In *2023 ieee international conference on robotics and automation (icra)*, pages 5954–5961. IEEE.

Atkeson, C. G., Moore, A. W., and Schaal, S. (1997). Locally weighted learning. *Lazy learning*, pages 11–73.

Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer.

Azuma, D., Miyanishi, T., Kurita, S., and Kawanabe, M. (2022). Scanqa: 3d question answering for spatial scene understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Bahl, S., Gupta, A., and Pathak, D. (2022). Human-to-robot imitation in the wild. *Robotics: Science and Systems (RSS)*.

Bahl, S., Mendonca, R., Chen, L., Jain, U., and Pathak, D. (2023). Affordances from human videos as a versatile representation for robotics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13778–13790.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Bar-Tal, O., Chefer, H., Tov, O., Herrmann, C., Paiss, R., Zada, S., Ephrat, A., Hur, J., Li, Y., Michaeli, T., et al. (2024). Lumiere: A space-time diffusion model for video generation. *arXiv preprint arXiv:2401.12945*.

Bardes, A., Ponce, J., and LeCun, Y. (2021). Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *arXiv preprint arXiv:2105.04906*.

Baruch, G., Chen, Z., Dehghan, A., Dimry, T., Feigin, Y., Fu, P., Gebauer, T., Joffe, B., Kurz, D., Schwartz, A., et al. (2021). Arkitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data. *arXiv preprint arXiv:2111.08897*.

Batra, D., Gokaslan, A., Kembhavi, A., Maksymets, O., Mottaghi, R., Savva, M., Toshev, A., and Wijmans, E. (2020). Objectnav revisited: On evaluation of embodied agents navigating to objects. *CoRR*, abs/2006.13171.

Bellman, R., Glicksberg, I., and Gross, O. (1956). On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 14(1):11–18.

Bescos, B., Campos, C., Tardós, J. D., and Neira, J. (2021). Dynaslam ii: Tightly-coupled multi-object tracking and slam. *IEEE robotics and automation letters*, 6(3):5191–5198.

Bescos, B., Fácil, J. M., Civera, J., and Neira, J. (2018). Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083.

Bharadhwaj, H., Vakil, J., Sharma, M., Gupta, A., Tulsiani, S., and Kumar, V. (2023). Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. *arXiv preprint arXiv:2309.01918*.

Bhattacharjee, T., Clever, H. M., Wade, J., and Kemp, C. C. (2018). Multimodal tactile perception of objects in a real home. *IEEE Robotics and Automation Letters*, 3(3):2523–2530.

Bhattacharjee, T., Wade, J., Chitalia, Y., and Kemp, C. C. (2016). Data-driven thermal recognition of contact with people and objects. In *2016 IEEE Haptics Symposium (HAPTICS)*, pages 297–304. IEEE.

Bhirangi, R., Hellebrekers, T., Majidi, C., and Gupta, A. (2021). Reskin: versatile, replaceable, lasting tactile skins. *arXiv preprint arXiv:2111.00071*.

Bhirangi, R., Pattabiraman, V., Erciyes, E., Cao, Y., Hellebrekers, T., and Pinto, L. (2024). Anyskin: Plug-and-play skin sensing for robotic touch.

Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2008). Survey: Robot programming by demonstration. *Handbook of robotics*, 59(BOOK_CHAP).

Bishop, C. M. (1994). Mixture density networks. *Neural Computing Research Group Report*.

Blukis, V., Paxton, C., Fox, D., Garg, A., and Artzi, Y. (2022). A persistent spatial semantic representation for high-level natural language instruction execution. In *Conference on Robot Learning*, pages 706–717. PMLR.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Bolte, B., Wang, A., Yang, J., Mukadam, M., Kalakrishnan, M., and Paxton, C. (2023). Usa-net: Unified semantic and affordance representations for robot memory. *arXiv preprint arXiv:2304.12164*.

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *ICRA*, pages 4243–4250.

Bowman, S. L., Atanasov, N., Daniilidis, K., and Pappas, G. J. (2017). Probabilistic data association for semantic slam. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1722–1729. IEEE.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Brahmbhatt, S., Ham, C., Kemp, C., and Hays, J. (2019). Contactdb: Analyzing and predicting grasp contact via thermal imaging.

Brandfonbrener, D., Bietti, A., Buckman, J., Laroche, R., and Bruna, J. (2022). When does return-conditioned supervised learning work for offline reinforcement learning? *arXiv preprint arXiv: Arxiv-2206.01079*.

Brandfonbrener, D., Nachum, O., and Bruna, J. (2023). Inverse dynamics pretraining learns good representations for multitask imitation. *arXiv preprint arXiv:2305.16985*.

Brasch, N., Bozic, A., Lallemand, J., and Tombari, F. (2018). Semantic monocular slam for highly dynamic environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 393–400. IEEE.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jackson, T., Jesmonth, S., Joshi, N. J., Julian, R., Kalashnikov, D., Kuang, Y., Leal, I., Lee, K.-H., Levine, S., Lu, Y., Malla, U., Manjunath, D., Mordatch, I., Nachum, O., Parada, C., Peralta, J., Perez, E., Pertsch, K., Quiambao, J., Rao, K., Ryoo, M., Salazar, G., Sanketi, P., Sayed, K., Singh, J., Sontakke, S., Stone, A., Tan, C., Tran, H., Vanhoucke, V., Vega, S., Vuong, Q., Xia, F., Xiao, T., Xu, P., Xu, S., Yu, T., and Zitkovich, B. (2023a). Rt-1: Robotics transformer for real-world control at scale.

Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. (2023b). Do as I can, not as I say: Grounding language in robotic affordances. In *CoRL*, pages 287–318. PMLR.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 33:1877–1901.

Bushaw, D. W. (1952). *Differential equations with a discontinuous forcing term.* PhD thesis, Princeton University.

Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., Zolna, K., Aytar, Y., Budden, D., Vecerik, M., et al. (2019). Scaling data-driven robotics with reward sketching and batch reinforcement learning. *arXiv preprint arXiv:1909.12200*.

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2020). nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*.

Carper, S. (2019). *Robots in American popular culture*. McFarland.

Chang, M., Gervet, T., Khanna, M., Yenamandra, S., Shah, D., Min, S. Y., Shah, K., Paxton, C., Gupta, S., Batra, D., Mottaghi, R., Malik, J., and Chaplot, D. S. (2023). Goat: Go to any thing.

Chaplot, D. S., Dalal, M., Gupta, S., Malik, J., and Salakhutdinov, R. R. (2021). Seal: Self-supervised embodied active learning using exploration and 3d consistency. *Advances in Neural Information Processing Systems*, 34:13086–13098.

Chaplot, D. S., Gandhi, D., Gupta, A., and Salakhutdinov, R. (2020). Object goal navigation using goal-oriented semantic exploration. In *In Neural Information Processing Systems (NeurIPS)*, volume 33, pages 4247–4258.

Chen, B., Sax, A., Lewis, G., Armeni, I., Savarese, S., Zamir, A., Malik, J., and Pinto, L. (2020a). Robust policies via mid-level visual representations: An experimental study in manipulation and navigation. *arXiv preprint arXiv:2011.06698*.

Chen, B., Xia, F., Ichter, B., Rao, K., Gopalakrishnan, K., Ryoo, M. S., Stone, A., and Kappler, D. (2022a). Open-vocabulary queryable scene representations for real world planning. In *arXiv preprint arXiv:2209.09874*.

Chen, D. Z., Chang, A. X., and Nießner, M. (2020b). Scanrefer: 3d object localization in rgb-d scans using natural language. *16th European Conference on Computer Vision (ECCV)*.

Chen, L., Bahl, S., and Pathak, D. (2023). Playfusion: Skill acquisition via diffusion from language-annotated play. In *Conference on Robot Learning*, pages 2012–2029. PMLR.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. (2020c). Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*.

Chen, X., Fan, H., Girshick, R., and He, K. (2020d). Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*.

Chen, Y.-C., Murali, A., Sundaralingam, B., Yang, W., Garg, A., and Fox, D. (2022b). Neural motion fields: Encoding grasp trajectories as implicit value functions. *arXiv preprint arXiv:2206.14854*.

Cheng, X., Li, J., Yang, S., Yang, G., and Wang, X. (2024). Open-television: Teleoperation with immersive active visual feedback.

Cheng, X., Shi, K., Agarwal, A., and Pathak, D. (2023). Extreme parkour with legged robots. *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11443–11450.

Chi, C., Feng, S., Du, Y., Xu, Z., Cousineau, E., Burchfiel, B., and Song, S. (2023). Diffusion policy: Visuomotor policy learning via action diffusion. In *RSS*.

Chi, C., Xu, Z., Pan, C., Cousineau, E., Burchfiel, B., Feng, S., Tedrake, R., and Song, S. (2024). Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots.

Clever, H. M., Handa, A., Mazhar, H., Parker, K., Shapira, O., Wan, Q., Narang, Y., Akinola, I., Cakmak, M., and Fox, D. (2021). Assistive tele-op: Leveraging transformers to collect robotic task demonstrations. *arXiv preprint arXiv:2112.05129*.

Codevilla, F., Santana, E., López, A. M., and Gaidon, A. (2019). Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338.

Cohen, N., Gal, R., Meirom, E. A., Chechik, G., and Atzmon, Y. (2022). " this is my unicorn, fluffy": Personalizing frozen vision-language representations. *arXiv preprint arXiv:2204.01694*.

Collins, J. A., Houff, C., Grady, P., and Kemp, C. C. (2023a). Visual contact pressure estimation for grippers in the wild. *arXiv preprint arXiv:2303.07344*.

Collins, J. A., Houff, C., Tan, Y. L., and Kemp, C. C. (2023b). Forcesight: Text-guided mobile manipulation with visual-force goals. *arXiv preprint arXiv:2309.12312*.

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub Repository*.

Cui, L. and Ma, C. (2019). Sof-slam: A semantic visual slam for dynamic environments. *IEEE access*, 7:166528–166539.

Cui, Z. J., Wang, Y., Shafiullah, N. M. M., and Pinto, L. (2022). From play to policy: Conditional behavior generation from uncurated robot data.

Dadashi, R., Hussenot, L., Vincent, D., Girgin, S., Raichuk, A., Geist, M., and Pietquin, O. (2021). Continuous control with action quantization from demonstrations. *arXiv preprint arXiv:2110.10149*.

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes.

Dalal, M., Chiruvolu, T., Chaplot, D., and Salakhutdinov, R. (2024). Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks. *arXiv preprint arXiv:2405.01534*.

Damen, D., Doughty, H., Farinella, G. M., Fidler, S., Furnari, A., Kazakos, E., Moltisanti, D., Munro, J., Perrett, T., Price, W., et al. (2018). Scaling egocentric vision: The epic-kitchens dataset. In *Proceedings of the European conference on computer vision (ECCV)*, pages 720–736.

Dasari, S., Ebert, F., Tian, S., Nair, S., Bucher, B., Schmeckpeper, K., Singh, S., Levine, S., and Finn, C. (2019). RoboNet: Large-scale multi-robot learning. In *Conference on Robot Learning (CoRL)*, volume 100, pages 885–897. PMLR.

Dasari, S. and Gupta, A. (2020). Transformers for one-shot visual imitation. *arXiv preprint arXiv:2011.05970*.

Dasari, S., Gupta, A., and Kumar, V. (2023). Learning dexterous manipulation from exemplar object trajectories and pre-grasps.

Datta, S., Dharur, S., Cartillier, V., Desai, R., Khanna, M., Batra, D., and Parikh, D. (2022). Episodic memory question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19119–19128.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., Zhang, X., Yu, X., Wu, Y., Wu, Z. F., Gou, Z., Shao, Z., Li, Z., Gao, Z., Liu, A., Xue, B., Wang, B., Wu, B., Feng, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Ding, H., Xin, H., Gao, H., Qu, H., Li, H., Guo, J., Li, J., Wang, J., Chen, J., Yuan, J., Qiu, J., Li, J., Cai, J. L., Ni, J., Liang, J., Chen, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Zhao, L., Wang, L., Zhang, L., Xu, L., Xia, L., Zhang, M., Zhang, M., Tang, M., Li, M., Wang, M., Li, M., Tian, N., Huang, P., Zhang, P., Wang, Q., Chen, Q., Du, Q., Ge, R., Zhang, R., Pan, R., Wang, R., Chen, R. J., Jin, R. L., Chen, R., Lu, S., Zhou, S., Chen, S., Ye, S., Wang, S., Yu, S., Zhou, S., Pan, S., Li, S. S., Zhou, S., Wu, S., Ye, S., Yun, T., Pei, T., Sun, T., Wang, T., Zeng, W., Zhao, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Xiao, W. L., An, W., Liu, X., Wang, X., Chen, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yang, X., Li, X., Su, X., Lin, X., Li, X. Q., Jin, X., Shen,

X., Chen, X., Sun, X., Wang, X., Song, X., Zhou, X., Wang, X., Shan, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhang, Y., Xu, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Yu, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Ou, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Zhu, Y. X., Xu, Y., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Tang, Y., Zha, Y., Yan, Y., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Ma, Z., Yan, Z., Wu, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Pan, Z., Huang, Z., Xu, Z., Zhang, Z., and Zhang, Z. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

Deitke, M., Batra, D., Bisk, Y., Campari, T., Chang, A. X., Chaplot, D. S., Chen, C., D'Arpino, C. P., Ehsani, K., Farhadi, A., et al. (2022). Retrospectives on the embodied ai workshop. *arXiv preprint arXiv:2210.06849*.

Deitke, M., Schwenk, D., Salvador, J., Weihs, L., Michel, O., VanderBilt, E., Schmidt, L., Ehsani, K., Kembhavi, A., and Farhadi, A. (2023). Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13142–13153.

Dempsey, P. (2023). Reviews-consumer technology. the teardown-amazon astro consumer robot. *Engineering & Technology*, 18(2):70–71.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255.

Depierre, A., Dellandréa, E., and Chen, L. (2018). Jacquard: A large scale dataset for robotic grasp detection. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3511–3516. IEEE.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, pages 4171–4186.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I. (2020). Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016). Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*.

Doersch, C., Gupta, A., and Efros, A. A. (2016). Unsupervised visual representation learning by context prediction.

Doshi, R., Walke, H., Mees, O., Dasari, S., and Levine, S. (2024). Scaling cross-embodied learning: One policy for manipulation, navigation, locomotion and aviation. *arXiv preprint arXiv:2408.11812*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dosovitskiy, A., Fischer, P., Springenberg, J. T., Riedmiller, M., and Brox, T. (2015). Discriminative unsupervised feature learning with exemplar convolutional neural networks.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR.

Driess, D., Huang, Z., Li, Y., Tedrake, R., and Toussaint, M. (2022). Learning multi-object dynamics with compositional neural radiance fields. *arXiv preprint arXiv:2202.11855*.

Du, Y., Ho, D., Alemi, A., Jang, E., and Khansari, M. (2022). Bayesian imitation learning for end-to-end mobile manipulation. In *International Conference on Machine Learning*, pages 5531–5546. PMLR.

Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. (2023). Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning*, pages 8657–8677. PMLR.

Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. In *Advances in neural information processing systems*, volume 30, pages 1087–1098.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1329–1338. JMLR.org.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110.

Dwibedi, D., Aytar, Y., Tompson, J., Sermanet, P., and Zisserman, A. (2021). With a little help from my friends: Nearest-neighbor contrastive learning of visual representations. *arXiv preprint arXiv:2104.14548*.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. (2018). Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*.

Ebert, F., Yang, Y., Schmeckpeper, K., Bucher, B., Georgakis, G., Daniilidis, K., Finn, C., and Levine, S. (2022). Bridge data: Boosting generalization of robotic skills with cross-domain datasets. In *Robotics: Science and Systems (RSS) XVIII*.

Ehsani, K., Gupta, T., Hendrix, R., Salvador, J., Weihs, L., Zeng, K.-H., Singh, K. P., Kim, Y., Han,

W., Herrasti, A., et al. (2023). Imitating shortest paths in simulation enables effective navigation and manipulation in the real world. *arXiv preprint arXiv:2312.02976*.

Emmons, S., Eysenbach, B., Kostrikov, I., and Levine, S. (2021). Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*.

Eppner, C., Mousavian, A., and Fox, D. (2021). Acronym: A large-scale grasp dataset based on simulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6222–6227. IEEE.

Etukuru, H., Naka, N., Hu, Z., Lee, S., Mehu, J., Edsinger, A., Paxton, C., Chintala, S., Pinto, L., and Shafiullah, N. M. M. (2024). Robot utility models: General policies for zero-shot deployment in new environments. *arXiv preprint arXiv:2409.05865*.

Eysenbach, B., Zhang, T., Salakhutdinov, R., and Levine, S. (2022). Contrastive learning as goal-conditioned reinforcement learning. *arXiv preprint arXiv: Arxiv-2206.07568*.

Falck, F., Larppichet, K., and Kormushev, P. (2019). De vito: A dual-arm, high degree-of-freedom, lightweight, inexpensive, passive upper-limb exoskeleton for robot teleoperation. In *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019, London, UK, July 3–5, 2019, Proceedings, Part I 20*, pages 78–89. Springer.

Fang, H., Fang, H.-S., Wang, Y., Ren, J., Chen, J., Zhang, R., Wang, W., and Lu, C. (2023a). Low-cost exoskeletons for learning whole-arm manipulation in the wild. *arXiv preprint arXiv:2309.14975*.

Fang, H.-S., Fang, H., Tang, Z., Liu, J., Wang, J., Zhu, H., and Lu, C. (2023b). RH20T: A robotic dataset for learning diverse skills in one-shot. In *RSS 2023 Workshop on Learning for Task and Motion Planning*.

Fang, H.-S., Wang, C., Fang, H., Gou, M., Liu, J., Yan, H., Liu, W., Xie, Y., and Lu, C. (2023c). Anygrasp:

Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*.

Fang, H.-S., Wang, C., Gou, M., and Lu, C. (2020). Graspnet-1billion: a large-scale benchmark for general object grasping. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11444–11453.

Finn, C. and Levine, S. (2017). Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE.

Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR.

FIRST (For Inspiration and Recognition of Science and Technology) (2024). FIRST robotics competition. Retrieved May 24, 2025, from https://www.firstinspires.org/robotics/frc.

Florence, P., Lynch, C., Zeng, A., Ramirez, O. A., Wahid, A., Downs, L., Wong, A., Lee, J., Mordatch, I., and Tompson, J. (2022). Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR.

Florence, P., Manuelli, L., and Tedrake, R. (2019). Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499.

Fried, D., Hu, R., Cirik, V., Rohrbach, A., Andreas, J., Morency, L., Berg-Kirkpatrick, T., Saenko, K., Klein, D., and Darrell, T. (2018). Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 3318–3329.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2020). D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Fu, Z., Cheng, X., and Pathak, D. (2022). Deep whole-body control: Learning a unified policy for manipulation and locomotion. *ArXiv*, abs/2210.10044.

Fu, Z., Zhao, Q., Wu, Q., Wetzstein, G., and Finn, C. (2024a). Humanplus: Humanoid shadowing and imitation from humans. *arXiv preprint arXiv:2406.10454*.

Fu, Z., Zhao, T. Z., and Finn, C. (2024b). Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. In *arXiv*.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, PMLR.

Gadre, S. Y., Ehsani, K., Song, S., and Mottaghi, R. (2022). Continuous scene representations for embodied ai. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14849–14859.

Gao, J., Sarkar, B., Xia, F., Xiao, T., Wu, J., Ichter, B., Majumdar, A., and Sadigh, D. (2024). Physically grounded vision-language models for robotic manipulation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12462–12469. IEEE.

Garrido, S., Moreno, L., Abderrahim, M., and Martin, F. (2006). Path planning for mobile robot navigation using voronoi diagram and fast marching. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2376–2381. IEEE.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

Gervet, T., Chintala, S., Batra, D., Malik, J., and Chaplot, D. S. (2023a). Navigating to objects in the real world. *Science Robotics*, 8(79):eadf6991.

Gervet, T., Xian, Z., Gkanatsios, N., and Fragkiadaki, K. (2023b). Act3d: 3d feature field transformers for multi-task robotic manipulation. In *Conference on Robot Learning*, pages 3949–3965. PMLR.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. (2019). Learning to reach goals via iterated supervised learning. *arXiv e-prints*, pages arXiv–1912.

Gidaris, S., Singh, P., and Komodakis, N. (2018). Unsupervised representation learning by predicting image rotations.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Gkioxari, G., Hariharan, B., Girshick, R. B., and Malik, J. (2014). R-cnns for pose estimation and action detection. *CoRR*, abs/1406.5212.

Google, G. T. (2024). Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.

Goyal, A., Friesen, A., Banino, A., Weber, T., Ke, N. R., Badia, A. P., Guez, A., Mirza, M., Humphreys, P. C., Konyushova, K., et al. (2022a). Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning*, pages 7740–7765. PMLR.

Goyal, A., Mousavian, A., Paxton, C., Chao, Y.-W., Okorn, B., Deng, J., and Fox, D. (2022b). Ifor: Iterative flow minimization for robotic object rearrangement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14787–14797.

Goyal, P., Niekum, S., and Mooney, R. (2021). Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. In *Conference on Robot Learning*, pages 485–497. PMLR.

Grady, P., Collins, J. A., Brahmbhatt, S., Twigg, C. D., Tang, C., Hays, J., and Kemp, C. C. (2022). Visual pressure estimation and control for soft robotic grippers. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3628–3635. IEEE.

Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al. (2021). Ego4d: Around the world in 3,000 hours of egocentric video. *arXiv preprint arXiv:2110.07058*, 3.

Grauman, K., Westbury, A., Byrne, E., Chavis, Z., Furnari, A., Girdhar, R., Hamburger, J., Jiang, H., Liu, M., Liu, X., et al. (2022). Ego4d: Around the world in 3,000 hours of egocentric video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18995–19012.

Grill, J.-B., Strub, F., Altche, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., et al. (2020). Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 33:21271–21284.

Gu, Q., Kuwajerwala, A., Morin, S., Jatavallabhula, K. M., Sen, B., Agarwal, A., Rivera, C., Paul, W., Ellis, K., Chellappa, R., et al. (2023). Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. *arXiv preprint arXiv:2309.16650*.

Gu, Q., Kuwajerwala, A., Morin, S., Jatavallabhula, K. M., Sen, B., Agarwal, A., Rivera, C., Paul, W., Ellis, K., Chellappa, R., et al. (2024). Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028. IEEE.

Gulcehre, C., Wang, Z., Novikov, A., Le Paine, T., Gomez Colmenarejo, S., Zolna, K., Agarwal,

R., Merel, J., Mankowitz, D., Paduraru, C., et al. (2020). Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv e-prints*, pages arXiv–2006.

Guo, Y., Wang, Y.-J., Zha, L., Jiang, Z., and Chen, J. (2023). Doremi: Grounding language model by detecting and recovering from plan-execution misalignment. *arXiv preprint arXiv:2307.00329*.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2019). Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*.

Gupta, A., Lynch, C., Kinman, B., Peake, G., Levine, S., and Hausman, K. (2022). Bootstrapped autonomous practicing via multi-task reinforcement learning. *arXiv preprint arXiv:2203.15755*.

Gupta, A., Murali, A., Gandhi, D. P., and Pinto, L. (2018). Robot learning in homes: Improving generalization and reducing dataset bias. In *Advances in Neural Information Processing Systems*, volume 31, pages 9094–9104.

Gupta, A., Zhang, M., Sathua, R., and Gupta, S. (2024). Opening cabinets and drawers in the real world using a commodity mobile manipulator. *arXiv preprint arXiv:2402.17767*.

Guzey, I., Dai, Y., Evans, B., Chintala, S., and Pinto, L. (2023a). See to touch: Learning tactile dexterity through visual incentives. *arXiv preprint arXiv:2309.12300*.

Guzey, I., Evans, B., Chintala, S., and Pinto, L. (2023b). Dexterity from touch: Self-supervised pre-training of tactile representations with robotic play. *arXiv preprint arXiv:2303.12076*.

Ha, H. and Song, S. (2022). Semantic abstraction: Open-world 3d scene understanding from 2d vision-language models. *arXiv preprint arXiv:2207.11514*.

Hahn, M., Chaplot, D. S., Tulsiani, S., Mukadam, M., Rehg, J. M., and Gupta, A. (2021). No rl, no simulation: Learning to navigate without navigating. *Advances in Neural Information Processing Systems*, 34:26661–26673.

Haldar, S., Mathur, V., Yarats, D., and Pinto, L. (2023a). Watch and match: Supercharging imitation with regularized optimal transport. In *Conference on Robot Learning*, pages 32–43. PMLR.

Haldar, S., Pari, J., Rai, A., and Pinto, L. (2023b). Teach a robot to fish: Versatile imitation from one minute of demonstrations. *arXiv preprint arXiv:2303.01497*.

Haldar, S., Peng, Z., and Pinto, L. (2024). Baku: An efficient transformer for multi-task policy learning.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. J. (2017). Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets. *Advances in neural information processing systems*, 30.

Haviland, J., Sünderhauf, N., and Corke, P. (2022). A holistic approach to reactive mobile manipulation. *IEEE Robotics and Automation Letters*, 7(2):3122–3129.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Henein, M., Zhang, J., Mahony, R., and Ila, V. (2020). Dynamic slam: The need for speed. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2123–2129. IEEE.

Henning, D. F., Laidlow, T., and Leutenegger, S. (2022). Bodyslam: Joint camera localisation, mapping, and human motion tracking. In *European Conference on Computer Vision*, pages 656–673. Springer.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2012). Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *International Journal of Robotic Research - IJRR*, 31:647–663.

Heo, M., Lee, Y., Lee, D., and Lim, J. J. (2023). Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*.

Herzog, A., Rao, K., Hausman, K., Lu, Y., Wohlhart, P., Yan, M., Lin, J., Arenas, M. G., Xiao, T., Kappler, D., et al. (2023). Deep rl at scale: Sorting waste in office buildings with a fleet of mobile manipulators. *arXiv preprint arXiv:2305.03270*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems*, volume 29, pages 4565–4573.

Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. In *NeurIPS*, volume 33, pages 6840–6851.

Hu, P., Huang, A., Dolan, J., Held, D., and Ramanan, D. (2021). Safe local motion planning with self-supervised freespace forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12732–12741.

Hu, S., Chen, L., Wu, P., Li, H., Yan, J., and Tao, D. (2022). St-p3: End-to-end vision-based autonomous driving via spatial-temporal feature learning. In *European Conference on Computer Vision*, pages 533–549. Springer.

Hu, Y., Yang, J., Chen, L., Li, K., Sima, C., Zhu, X., Chai, S., Du, S., Lin, T., Wang, W., et al. (2023). Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862.

Huang, C., Mees, O., Zeng, A., and Burgard, W. (2023a). Audio visual language maps for robot navigation. *arXiv preprint arXiv:2303.07522*.

Huang, C., Mees, O., Zeng, A., and Burgard, W. (2023b). Visual language maps for robot navigation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10608–10615. IEEE.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022a). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*, pages 9118–9147. PMLR.

Huang, W., Wang, C., Zhang, R., Li, Y., Wu, J., and Fei-Fei, L. (2023c). VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *CoRL*.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Brown, N., Jackson, T., Luu, L., Levine, S., Hausman, K., and Ichter, B. (2022b). Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

Ishiguro, Y., Makabe, T., Nagamatsu, Y., Kojio, Y., Kojima, K., Sugai, F., Kakiuchi, Y., Okada, K., and Inaba, M. (2020). Bilateral humanoid teleoperation system using whole-body exoskeleton cockpit tablis. *IEEE Robotics and Automation Letters*, 5(4):6419–6426.

Islam, R., Zang, H., Goyal, A., Lamb, A., Kawaguchi, K., Li, X., Laroche, R., Bengio, Y., and Combes, R. T. D. (2022). Discrete factorial representations as an abstraction for goal conditioned reinforcement learning. *arXiv preprint arXiv:2211.00247*.

Ivanovic, B., Schmerling, E., Leung, K., and Pavone, M. (2018). Generative modeling of multimodal multi-human behavior. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3088–3095. IEEE.

Iyer, A., Peng, Z., Dai, Y., Guzey, I., Haldar, S., Chintala, S., and Pinto, L. (2024). Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*.

Jain, A. and Kemp, C. C. (2013). Improving robot manipulation with data-driven object-centric models of everyday forces. *Autonomous Robots*, 35:143–159.

Jain, A., Nguyen, H., Rath, M., Okerman, J., and Kemp, C. C. (2010). The complex structure of simple devices: A survey of trajectories and forces that open doors and drawers. In *2010 3rd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics*, pages 184–190. IEEE.

Jang, E., Irpan, A., Khansari, M., Kappler, D., Ebert, F., Lynch, C., Levine, S., and Finn, C. (2021). BC-Z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning (CoRL)*, pages 991–1002. PMLR.

Janner, M., Li, Q., and Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34.

Jatavallabhula, K. M., Kuwajerwala, A., Gu, Q., Omama, M., Chen, T., Li, S., Iyer, G., Saryazdi, S., Keetha, N., Tewari, A., et al. (2023). Conceptfusion: Open-set multimodal 3d mapping. *arXiv preprint arXiv:2302.07241*.

Ji, M., Qiu, R.-Z., Zou, X., and Wang, X. (2024). Graspsplats: Efficient manipulation with 3d feature splatting. *arXiv preprint arXiv:2409.02084*.

Jiang, B., Chen, S., Xu, Q., Liao, B., Chen, J., Zhou, H., Zhang, Q., Liu, W., Huang, C., and Wang, X. (2023). Vad: Vectorized scene representation for efficient autonomous driving. *arXiv preprint arXiv:2303.12077*.

Jiang, Y., Moseson, S., and Saxena, A. (2011). Efficient grasping from RGBD images: Learning using a new rectangle representation. In *2011 IEEE International conference on robotics and automation*, pages 3304–3311. IEEE.

Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Jones, J. L. (2006). Robots at the tipping point: the road to irobot roomba. *IEEE Robotics & Automation Magazine*, 13(1):76–78.

Kaelbling, L. P. (1993). Learning to achieve goals. In *IN PROC. OF IJCAI-93*, pages 1094–1098. Morgan Kaufmann.

Kalakrishnan, M., Pastor, P., Righetti, L., and Schaal, S. (2013). Learning objective functions for manipulation. In *2013 IEEE International Conference on Robotics and Automation*, pages 1331–1336. IEEE.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.

Kalashnikov, D., Varley, J., Chebotar, Y., Swanson, B., Jonschkowski, R., Finn, C., Levine, S., and Hausman, K. (2021). Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Kappler, D., Bohg, J., and Schaal, S. (2015). Leveraging big data for grasp planning. In *ICRA*, pages 4304–4311.

Karamcheti, S., Nair, S., Chen, A. S., Kollar, T., Finn, C., Sadigh, D., and Liang, P. (2023). Language-driven representation learning for robotics. *Robotics: Science and Systems (RSS)*.

Karpathy, A. (2020). GitHub - karpathy/minGPT: A minimal PyTorch re-implementation of the OpenAI GPT (Generative Pretrained Transformer) training.

Kemp, C. C., Edsinger, A., Clever, H. M., and Matulevich, B. (2022). The design of stretch: A compact, lightweight mobile manipulator for indoor human environments. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3150–3157. IEEE.

Kerbl, B., Kopanas, G., Leimkühler, T., and Drettakis, G. (2023). 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4).

Kerr, J., Kim, C. M., Goldberg, K., Kanazawa, A., and Tancik, M. (2023). Lerf: Language embedded radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19729–19739.

Khazatsky, A., Pertsch, K., Nair, S., Balakrishna, A., Dasari, S., Karamcheti, S., Nasiriany, S., Srirama, M. K., Chen, L. Y., Ellis, K., et al. (2024). Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*.

Khurana, T., Hu, P., Dave, A., Ziglar, J., Held, D., and Ramanan, D. (2022). Differentiable raycasting for self-supervised occupancy forecasting. In *European Conference on Computer Vision*, pages 353–369. Springer.

Kim, J., hyeon Park, J., Cho, D., and Kim, H. J. (2022). Automating reinforcement learning with example-based resets. *IEEE Robotics and Automation Letters*, 7(3):6606–6613.

Kim, M. J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E., Lam, G., Sanketi, P., et al. (2024). Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W.-Y., Dollar, P., and Girshick, R. (2023). Segment anything. In *ICCV*, pages 4015–4026.

Kobayashi, S., Matsumoto, E., and Sitzmann, V. (2022). Decomposing nerf for editing via feature field distillation. *arXiv preprint arXiv:2205.15585*.

Kohlbrecher, S., Meyer, J., von Stryk, O., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE.

Kostrikov, I., Tompson, J., Fergus, R., and Nachum, O. (2021). Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*.

Krantz, J., Lee, S., Malik, J., Batra, D., and Chaplot, D. S. (2022). Instance-specific image goal navigation: Training embodied agents to find object instances. *arXiv preprint arXiv:2211.15876*.

Krishna, G. S., Supriya, K., and Baidya, S. (2023). 3ds-slam: A 3d object detection based semantic slam towards dynamic indoor environments. *arXiv preprint arXiv:2310.06385*.

Ku, A., Anderson, P., Patel, R., Ie, E., and Baldridge, J. (2020). Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4392–4412, Online. Association for Computational Linguistics.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32:11761–11771.

Kumar, A., Singh, A., Ebert, F., Nakamoto, M., Yang, Y., Finn, C., and Levine, S. (2022). Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *arXiv preprint arXiv:2210.05178*.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191.

Lambert, N., Morrison, J., Pyatkin, V., Huang, S., Ivison, H., Brahman, F., Miranda, L. J. V., Liu, A., Dziri, N., Lyu, S., Gu, Y., Malik, S., Graf, V., Hwang, J. D., Yang, J., Bras, R. L., Tafjord, O., Wilhelm, C., Soldaini, L., Smith, N. A., Wang, Y., Dasigi, P., and Hajishirzi, H. (2025). Tulu 3: Pushing frontiers in open language model post-training.

Lee, M. and Anderson, C. W. (2016). Robust reinforcement learning with relevance vector machines. *Robot Learning and Planning (RLP 2016)*, page 5.

Lee, N. and Kitani, K. M. (2016). Predicting wide receiver trajectories in american football. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9. IEEE.

Lee, S., Wang, Y., Etukuru, H., Kim, H. J., Shafiullah, N. M. M., and Pinto, L. (2024). Behavior generation with latent actions. *arXiv preprint arXiv:2403.03181*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *JMLR*, 17(1):1334–1373.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436.

Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2017). Learning multi-level hierarchies with hindsight. *arXiv preprint arXiv:1712.00948*.

Li, X., Liu, M., Zhang, H., Yu, C., Xu, J., Wu, H., Cheang, C., Jing, Y., Zhang, W., Liu, H., et al. (2023). Vision-language foundation models as effective robot imitators. *arXiv preprint arXiv:2311.01378*.

Li, Y., Li, S., Sitzmann, V., Agrawal, P., and Torralba, A. (2022). 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*, pages 112–123. PMLR.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., and Zeng, A. (2023). Code as Policies: Language model programs for embodied control. In *icra*, pages 9493–9500. IEEE.

Lin, T., Zhang, Y., Li, Q., Qi, H., Yi, B., Levine, S., and Malik, J. (2024). Learning visuotactile skills with two multifingered hands. *arXiv preprint arXiv:2404.16823*.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Liu, B., Zhu, Y., Gao, C., Feng, Y., Liu, Q., Zhu, Y., and Stone, P. (2024a). Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. (2023a). Visual instruction tuning.

Liu, M., He, T., Xu, M., and Zhang, W. (2020). Energy-based imitation learning. *arXiv preprint arXiv:2004.09395*, 33.

Liu, P., Guo, Z., Warke, M., Chintala, S., Paxton, C., Shafiullah, N. M. M., and Pinto, L. (2024b). Dynamem: Online dynamic spatio-semantic memory for open world mobile manipulation.

Liu, P., Orru, Y., Vakil, J., Paxton, C., Shafiullah, N., and Pinto, L. (2024c). Demonstrating ok-robot: What really matters in integrating open-knowledge models for robotics. In *Robotics: Science and Systems XX*, RSS2024. Robotics: Science and Systems Foundation.

Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., and Zhang, L. (2023b). Grounding dino: Marrying dino with grounded pre-training for open-set object detection.

Liu, W., Hermans, T., Chernova, S., and Paxton, C. (2022). Structdiffusion: Object-centric diffusion for semantic rearrangement of novel objects. *arXiv preprint arXiv:2211.04604*.

Liu, Z., Bahety, A., and Song, S. (2023c). Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2018). Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11.

Luo, J., Dong, P., Wu, J., Kumar, A., Geng, X., and Levine, S. (2023). Action-quantized offline reinforcement learning for robotic skill learning. In *Conference on Robot Learning*, pages 1348–1361. PMLR.

Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. (2020). Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132. PMLR.

Lynch, C. and Sermanet, P. (2021). Language conditioned imitation learning over unstructured data. *Robotics: Science and Systems*.

Lynch, C., Wahid, A., Tompson, J., Ding, T., Betker, J., Baruch, R., Armstrong, T., and Florence, P. (2023). Interactive language: Talking to robots in real time. *IEEE Robotics and Automation Letters*.

Ma, L., Stückler, J., Kerl, C., and Cremers, D. (2017). Multi-view deep learning for consistent semantic mapping with rgb-d cameras. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 598–605. IEEE.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*.

Ma, Y. J., Liang, W., Wang, H.-J., Wang, S., Zhu, Y., Fan, L., Bastani, O., and Jayaraman, D. (2024). Dreureka: Language model guided sim-to-real transfer. *arXiv preprint arXiv:2406.01967*.

Ma, Y. J., Sodhani, S., Jayaraman, D., Bastani, O., Kumar, V., and Zhang, A. (2022a). Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*.

Ma, Y. J., Yan, J., Jayaraman, D., and Bastani, O. (2022b). How far i'll go: Offline goal-conditioned reinforcement learning via f-advantage regression. *arXiv preprint arXiv:2206.03023*.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

Maggio, D., Abate, M., Shi, J., Mario, C., and Carlone, L. (2023). Loc-nerf: Monte carlo localization using neural radiance fields. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4018–4025. IEEE.

Maggio, D., Chang, Y., Hughes, N., Trang, M., Griffith, D., Dougherty, C., Cristofalo, E., Schmid, L., and Carlone, L. (2024). Clio: Real-time task-driven open-set 3d scene graphs. *arXiv preprint arXiv:2404.13696*.

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. (2017a). Dex-Net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Robotics: Science and Systems (RSS)*.

Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. (2017b). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*.

Mahler, J., Matl, M., Liu, X., Li, A., Gealy, D., and Goldberg, K. (2018). Dex-net 3.0: Computing robust robot vacuum suction grasp targets in point clouds using a new analytic model and deep learning.

Majumdar, A., Ajay, A., Zhang, X., Putta, P., Yenamandra, S., Henaff, M., Silwal, S., Mcvay, P., Maksymets, O., Arnaud, S., et al. (2024). Openeqa: Embodied question answering in the era of foundation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16488–16498.

Majumdar, A., Shrivastava, A., Lee, S., Anderson, P., Parikh, D., and Batra, D. (2020). Improving vision-and-language navigation with image-text pairs from the web. In *ECCV*, pages 259–274. Springer.

Majumdar, A., Yadav, K., Arnaud, S., Ma, Y. J., Chen, C., Silwal, S., Jain, A., Berges, V.-P., Abbeel, P., Malik, J., et al. (2023). Where are we in the search for an artificial visual cortex for embodied intelligence? *arXiv preprint arXiv:2303.18240*.

Mandi, Z., Liu, F., Lee, K., and Abbeel, P. (2021). Towards more generalizable one-shot visual imitation learning. *arXiv preprint arXiv:2110.13423*.

Mandlekar, A., Booher, J., Spero, M., Tung, A., Gupta, A., Zhu, Y., Garg, A., Savarese, S., and Fei-Fei, L. (2019). Scaling robot supervision to hundreds of hours with RoboTurk: Robotic manipulation dataset through human reasoning and dexterity. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1048–1055. IEEE.

Mandlekar, A., Xu, D., Martín-Martín, R., Savarese, S., and Fei-Fei, L. (2020). Learning to Generalize Across Long-Horizon Tasks from Human Demonstrations. *arXiv e-prints*, page arXiv:2003.06085.

Mandlekar, A., Xu, D., Wong, J., Nasiriany, S., Wang, C., Kulkarni, R., Fei-Fei, L., Savarese, S., Zhu, Y., and Martín-Martín, R. (2021). What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*.

Mandlekar, A., Zhu, Y., Garg, A., Booher, J., Spero, M., Tung, A., Gao, J., Emmons, J., Gupta, A., Orbay, E., et al. (2018). Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, pages 879–893. PMLR.

Mansimov, E. and Cho, K. (2018). Simple nearest neighbor policy method for continuous control tasks.

Manuelli, L., Li, Y., Florence, P., and Tedrake, R. (2020). Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning. *arXiv preprint arXiv:2009.05085*.

Mao, J., Qian, Y., Zhao, H., and Wang, Y. (2023a). Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*.

Mao, J., Ye, J., Qian, Y., Pavone, M., and Wang, Y. (2023b). A language agent for autonomous driving. *arXiv preprint arXiv:2311.10813*.

Margolis, G., Yang, G., Paigwar, K., Chen, T., and Agrawal, P. (2022). Rapid locomotion via reinforcement learning. *The International Journal of Robotics Research*, 43:572 – 587.

Marino, K., Rastegari, M., Farhadi, A., and Mottaghi, R. (2019). Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 3195–3204.

Matsui, Y., Uchida, Y., Jégou, H., and Satoh, S. (2018). A survey of product quantization. *ITE Transactions on Media Technology and Applications*, 6(1):2–10.

Matsuki, H., Murai, R., Kelly, P. H., and Davison, A. J. (2024). Gaussian splatting slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18039–18048.

Mazzaglia, P., Verbelen, T., Dhoedt, B., Lacoste, A., and Rajeswar, S. (2022). Choreographer: Learning and adapting skills in imagination. *arXiv preprint arXiv:2211.13350*.

McCormac, J., Clark, R., Bloesch, M., Davison, A., and Leutenegger, S. (2018). Fusion++: Volumetric object-level slam. In *2018 international conference on 3D vision (3DV)*, pages 32–41. IEEE.

Medeiros, L. (2023). Lang segment anything. https://github.com/luca-medeiros/lang-segment-anything.

Melnik, A., Büttner, M., Harz, L., Brown, L., Nandi, G. C., PS, A., Yadav, G. K., Kala, R., and Haschke, R. (2023). Uniteam: Open vocabulary mobile manipulation challenge.

Meltzoff, A. N. and Moore, K. (1983). Newborn infants imitate adult facial gestures. *Child development*.

Meltzoff, A. N. and Moore, M. K. (1977). Imitation of facial and manual gestures by human neonates. *Science*.

Metz, L., Ibarz, J., Jaitly, N., and Davidson, J. (2017). Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*.

Michael, E., Summers, T., Wood, T. A., Manzie, C., and Shames, I. (2022). Probabilistic data association for semantic slam at scale. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4359–4364. IEEE.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. *European Conference on Computer Vision (ECCV)*, 65(1):99–106.

Min, S. Y., Chaplot, D. S., Ravikumar, P., Bisk, Y., and Salakhutdinov, R. (2021). Film: Following instructions in language with modular methods. *arXiv preprint arXiv:2110.07342*.

Minderer, M., Gritsenko, A., and Houlsby, N. (2024). Scaling open-vocabulary object detection.

Minderer, M., Gritsenko, A., Stone, A., Neumann, M., Weissenborn, D., Dosovitskiy, A., Mahendran, A., Arnab, A., Dehghani, M., Shen, Z., Wang, X., Zhai, X., Kipf, T., and Houlsby, N. (2022). Simple open-vocabulary object detection with vision transformers. In *European Conference on Computer Vision*, pages 728–755. Springer.

Misra, I., Zitnick, C. L., and Hebert, M. (2016). Shuffle and learn: Unsupervised learning using temporal order verification.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Mousavian, A., Eppner, C., and Fox, D. (2019a). 6-dof graspnet: Variational grasp generation for object manipulation.

Mousavian, A., Toshev, A., Fišer, M., Košecká, J., Wahid, A., and Davidson, J. (2019b). Visual representations for semantic target driven navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8846–8852. IEEE.

Mu, Y., Yao, S., Ding, M., Luo, P., and Gan, C. (2023). EC2: Emergent communication for embodied control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6704–6714.

Mullen Jr, J. F. and Manocha, D. (2024). Towards robots that know when they need help: Affordance-based uncertainty for large language model planners. *arXiv preprint arXiv:2403.13198*.

Müller, T., Evans, A., Schied, C., and Keller, A. (2022). Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15.

Nachum, O., Gu, S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on*

Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, volume 31, pages 3307–3317.

Nachum, O. and Yang, M. (2021). Provable representation learning for imitation with contrastive fourier features. *arXiv preprint arXiv:2105.12272*, 34:30100–30112.

Nair, S., Mitchell, E., Chen, K., Savarese, S., Finn, C., et al. (2022a). Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. (2022b). R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv: Arxiv-2203.12601*.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, volume 1, pages 663–670.

Nguyen, H. and Kemp, C. C. (2014). Autonomously learning to visually detect where manipulation will succeed. *Autonomous Robots*, 36:137–152.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

OpenAI (2023). GPT-4 technical report.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al. (2023). Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*.

Ortiz, J., Clegg, A., Dong, J., Sucar, E., Novotny, D., Zollhoefer, M., and Mukadam, M. (2022). isdf: Real-time neural signed distance fields for robot perception. *arXiv preprint arXiv:2204.02296*.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. (2018). An algorithmic perspective on imitation learning. *arXiv preprint arXiv:1811.06711*.

Ozyesil, O., Voroninski, V., Basri, R., and Singer, A. (2017). A survey of structure from motion.

Padalkar, A., Pooley, A., Jain, A., Bewley, A., Herzog, A., Irpan, A., Khazatsky, A., Rai, A., Singh, A., Brohan, A., et al. (2023). Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*.

Parashar, P., Vakil, J., Powers, S., and Paxton, C. (2023). Spatial-language attention policies for efficient robot learning. *arXiv preprint arXiv:2304.11235*.

Pari, J., Muhammad, N., Arunachalam, S. P., Pinto, L., et al. (2021). The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*.

Park, J., Lim, S., Lee, J., Park, S., Chang, M., Yu, Y., and Choi, S. (2023). Clara: classifying and disambiguating user commands for reliable interactive robotic agents. *IEEE Robotics and Automation Letters*.

Park, Y. and Agrawal, P. (2024). Using apple vision pro to train and control robots.

Paster, K., McIlraith, S., and Ba, J. (2022). You can't count on luck: Why decision transformers fail in stochastic environments. *arXiv preprint arXiv: Arxiv-2205.15967*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037.

Pattabiraman, V., Huang, Z., Panozzo, D., Zorin, D., Pinto, L., and Bhirangi, R. (2025). eflesh: Highly customizable magnetic touch sensing using cut-cell microstructures.

Pearce, T., Rashid, T., Kanervisto, A., Bignell, D., Sun, M., Georgescu, R., Macua, S. V., Tan, S. Z., Momennejad, I., Hofmann, K., et al. (2023). Imitating human behaviour with diffusion models. *arXiv preprint arXiv:2301.10677*.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14.

Peng, X. B., Ma, Z., Abbeel, P., Levine, S., and Kanazawa, A. (2021). Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (TOG)*, 40(4):1–20.

Pertsch, K., Lee, Y., and Lim, J. (2021). Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pages 188–204. PMLR.

Piaget, J. (2013). *Play, dreams and imitation in childhood*, volume 25. Routledge.

Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *ICRA*, pages 3406–3413.

Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., and Rombach, R. (2023). Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*.

Pomerleau, D. A. (1988). Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1.

Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313.

Pritzel, A., Uria, B., Srinivasan, S., Puigdomènech, A., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural episodic control.

Qin, Y., Chen, R., Zhu, H., Song, M., Xu, J., and Su, H. (2019). S4g: Amodal single-view single-shot se(3) grasp detection in cluttered scenes.

Qiu, R.-Z., Hu, Y., Song, Y., Yang, G., Fu, Y., Ye, J., Mu, J., Yang, R., Atanasov, N., Scherer, S., et al. (2024). Learning generalizable feature fields for mobile manipulation. *arXiv preprint arXiv:2403.07563*.

Qiu, Y., Wang, C., Wang, W., Henein, M., and Scherer, S. (2022). Airdos: Dynamic slam benefits from articulated objects. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8047–8053. IEEE.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision. In Meila, M. and Zhang, T., editors, *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Radosavovic, I., Xiao, T., James, S., Abbeel, P., Malik, J., and Darrell, T. (2022). Real-world robot learning with masked visual pre-training. In *Conference on Robot Learning*.

Rahmatizadeh, R., Abolghasemi, P., Bölöni, L., and Levine, S. (2018). Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3758–3765. IEEE.

Rainbow Robotics (2025). Rb-y1 mobile manipulator. Retrieved May 24, 2025, from https://rainbowrobotics.github.io/rby1-dev/.

Rajaraman, N., Yang, L., Jiao, J., and Ramchandran, K. (2020). Toward the fundamental limits of imitation learning. *Advances in Neural Information Processing Systems*, 33:2914–2924.

Rajeswaran, A., Lowrey, K., Todorov, E., and Kakade, S. (2018). Towards generalization and simplicity in continuous control.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. (2022). Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*.

Ramrakhya, R., Batra, D., Wijmans, E., and Das, A. (2023). Pirlnav: Pretraining with imitation and rl finetuning for objectnav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17896–17906.

Rana, K., Haviland, J., Garg, S., Abou-Chakra, J., Reid, I., and Suenderhauf, N. (2023). Sayplan: Grounding large language models using 3d scene graphs for scalable task planning. *arXiv preprint arXiv:2307.06135*.

Rashid, A., Sharma, S., Kim, C. M., Kerr, J., Chen, L. Y., Kanazawa, A., and Goldberg, K. (2023). Language embedded radiance fields for zero-shot task-oriented grasping. In *7th Annual Conference on Robot Learning*.

Rasmussen, C. E. and Nickisch, H. (2010). Gaussian processes for machine learning (gpml) toolbox. *The Journal of Machine Learning Research*, 11:3011–3015.

Ravi, N., Gabeur, V., Hu, Y.-T., Hu, R., Ryali, C., Ma, T., Khedr, H., Rädle, R., Rolland, C., Gustafson, L., Mintun, E., Pan, J., Alwala, K. V., Carion, N., Wu, C.-Y., Girshick, R., Dollár, P., and Feichtenhofer, C. (2024). Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-maron, G., Giménez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N.,

Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., and de Freitas, N. (2022). A generalist agent. *Transactions on Machine Learning Research*.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ren, A. Z., Dixit, A., Bodrova, A., Singh, S., Tu, S., Brown, N., Xu, P., Takayama, L., Xia, F., Varley, J., et al. (2023). Robots that ask for help: Uncertainty alignment for large language model planners. *arXiv preprint arXiv:2307.01928*.

Reuss, M., Li, M., Jia, X., and Lioutikov, R. (2023). Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*.

Reynolds, L. and McDonell, K. (2021). Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rosinol, A., Leonard, J. J., and Carlone, L. (2023). Nerf-slam: Real-time dense monocular slam with neural radiance fields. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3437–3444. IEEE.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured

prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

Rozenberszki, D., Litany, O., and Dai, A. (2022). Language-grounded indoor 3d semantic segmentation in the wild.

Rudin, N., Hoeller, D., Reist, P., and Hutter, M. (2022). Learning to walk in minutes using massively parallel deep reinforcement learning. In Faust, A., Hsu, D., and Neumann, G., editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 91–100. PMLR.

Russell, S. (1998). Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103.

Sax, A., Zhang, J. O., Emi, B., Zamir, A., Savarese, S., Guibas, L., and Malik, J. (2019). Learning to navigate using mid-level visual priors. *arXiv preprint arXiv:1912.11121*.

Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France. PMLR.

Schmid, L., Abate, M., Chang, Y., and Carlone, L. (2024). Khronos: A unified approach for spatio-temporal metric-semantic slam in dynamic environments. In *Proc. of Robotics: Science and Systems*.

Schonberger, J. L. and Frahm, J.-M. (2016). Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113.

Sermanet, P., Xu, K., and Levine, S. (2016). Unsupervised perceptual rewards for imitation learning. *arXiv preprint arXiv:1612.06699*.

Shafiullah, N. M., Cui, Z., Altanzaya, A. A., and Pinto, L. (2022). Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968.

Shafiullah, N. M., Paxton, C., Pinto, L., Chintala, S., and Szlam, A. (2023a). Clip-fields: Weakly supervised semantic fields for robotic memory. In *Robotics: Science and Systems XIX*. Robotics: Science and Systems Foundation.

Shafiullah, N. M. M., Rai, A., Etukuru, H., Liu, Y., Misra, I., Chintala, S., and Pinto, L. (2023b). On bringing robots home.

Shah, D. and Levine, S. (2022). Viking: Vision-based kilometer-scale navigation with geographic hints. *arXiv preprint arXiv:2202.11271*.

Shah, D., Sridhar, A., Dashora, N., Stachowicz, K., Black, K., Hirose, N., and Levine, S. (2023). ViNT: A Foundation Model for Visual Navigation. In *7th Annual Conference on Robot Learning (CoRL)*.

Shah, D. and Xie, Q. (2018). Q-learning with nearest neighbors.

Sharma, P., Mohan, L., Pinto, L., and Gupta, A. (2018). Multiple interactions made easy (mime): Large scale demonstrations data for imitation. *arXiv preprint arXiv:1810.07121*, pages 906–915.

Sharma, P., Torralba, A., and Andreas, J. (2021). Skill induction and planning with latent language. *arXiv preprint arXiv:2110.01517*.

Shen, W., Yang, G., Yu, A., Wong, J., Kaelbling, L. P., and Isola, P. (2023). Distilled feature fields enable few-shot language-guided manipulation. *arXiv preprint arXiv:2308.07931*.

Shi, W., Xu, J., Zhu, D., Zhang, G., Wang, X., Li, J., and Zhang, X. (2021). Rgb-d semantic segmentation and label-oriented voxelgrid fusion for accurate 3d semantic mapping. *IEEE transactions on circuits and systems for video technology*, 32(1):183–197.

Shorinwa, O., Tucker, J., Smith, A., Swann, A., Chen, T., Firoozi, R., Kennedy, M. D., and Schwager, M. (2024). Splat-mover: Multi-stage, open-vocabulary robotic manipulation via editable gaussian splatting. In *8th Annual Conference on Robot Learning*.

Shridhar, M., Manuelli, L., and Fox, D. (2022). Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR.

Shridhar, M., Manuelli, L., and Fox, D. (2023). Perceiver-Actor: A multi-task transformer for robotic manipulation. In *CoRL*, pages 785–799. PMLR.

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. (2020). Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749.

Sian, N. E., Yokoi, K., Kajita, S., Kanehiro, F., and Tanie, K. (2004). Whole body teleoperation of a humanoid robot development of a simple master device using joysticks. *Journal of the Robotics Society of Japan*, 22(4):519–527.

Simeonov, A., Du, Y., Tagliasacchi, A., Tenenbaum, J. B., Rodriguez, A., Agrawal, P., and Sitzmann, V. (2022). Neural descriptor fields: Se (3)-equivariant object representations for manipulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6394–6400. IEEE.

Singh, A., Liu, H., Zhou, G., Yu, A., Rhinehart, N., and Levine, S. (2020). Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., and Fox, D. (2023). Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, page 11523.

Sitzmann, V., Zollhöfer, M., and Wetzstein, G. (2019). Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems*, 32.

Snell, J., Swersky, K., and Zemel, R. S. (2017). Prototypical networks for few-shot learning.

Somasundaram, K., Dong, J., Tang, H., Straub, J., Yan, M., Goesele, M., Engel, J. J., De Nardi, R., and Newcombe, R. (2023). Project aria: A new tool for egocentric multi-modal ai research. *arXiv preprint arXiv:2308.13561*.

Song, C. H., Wu, J., Washington, C., Sadler, B. M., Chao, W.-L., and Su, Y. (2023). Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009.

Song, S., Lim, H., Lee, A. J., and Myung, H. (2022). Dynavins: a visual-inertial slam for dynamic environments. *IEEE Robotics and Automation Letters*, 7(4):11523–11530.

Song, S., Zeng, A., Lee, J., and Funkhouser, T. (2020). Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters*, 5(3):4978–4985.

Song, Z., Zhang, G., Xie, J., Liu, L., Jia, C., Xu, S., and Wang, Z. (2024). Voxelnextfusion: A simple, unified and effective voxel fusion framework for multi-modal 3d object detection. *arXiv preprint arXiv:2401.02702*.

Sridhar, A., Shah, D., Glossop, C., and Levine, S. (2024). Nomad: Goal masked diffusion policies for navigation and exploration. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70. IEEE.

Stadie, B. C., Abbeel, P., and Sutskever, I. (2017). Third-person imitation learning. *ICLR*.

Stolle, M. and Precup, D. (2002). Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer.

Stone, A., Xiao, T., Lu, Y., Gopalakrishnan, K., Lee, K.-H., Vuong, Q., Wohlhart, P., Kirmani, S., Zitkovich, B., Xia, F., Finn, C., and Hausman, K. (2023). Open-world object manipulation using pre-trained vision-language model. In *arXiv preprint*.

Stooke, A., Lee, K., Abbeel, P., and Laskin, M. (2021). Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pages 9870–9879. PMLR.

Sucar, E., Liu, S., Ortiz, J., and Davison, A. J. (2021). imap: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6229–6238.

Sundermeyer, M., Mousavian, A., Triebel, R., and Fox, D. (2021). Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13438–13444. IEEE.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

Tavakoli, A., Pardo, F., and Kormushev, P. (2018). Action branching architectures for deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 32.

Team, G. R., Abeyruwan, S., Ainslie, J., Alayrac, J.-B., Arenas, M. G., Armstrong, T., Balakrishna, A., Baruch, R., Bauza, M., Blokzijl, M., et al. (2025a). Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*.

Team, O. (2024). Gpt-4 technical report.

Team, O. M., Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Kreiman, T., Xu, C., et al. (2024). Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*.

Team, T. L., Barreiros, J., Beaulieu, A., Bhat, A., Cory, R., Cousineau, E., Dai, H., Fang, C.-H., Hashimoto, K., Irshad, M. Z., Itkina, M., Kuppuswamy, N., Lee, K.-H., Liu, K., McConachie, D., McMahon, I., Nishimura, H., Phillips-Grafflin, C., Richter, C., Shah, P., Srinivasan, K., Wulfe, B., Xu, C., Zhang, M., Alspach, A., Angeles, M., Arora, K., Guizilini, V. C., Castro, A., Chen, D., Chu, T.-S., Creasey, S., Curtis, S., Denitto, R., Dixon, E., Dusel, E., Ferreira, M., Goncalves, A., Gould, G., Guoy, D., Gupta, S., Han, X., Hatch, K., Hathaway, B., Henry, A., Hochsztein, H., Horgan, P., Iwase, S., Jackson, D., Karamcheti, S., Keh, S., Masterjohn, J., Mercat, J., Miller, P., Mitiguy, P., Nguyen, T., Nimmer, J., Noguchi, Y., Ong, R., Onol, A., Pfannenstiehl, O., Poyner, R., Rocha, L. P. M., Richardson, G., Rodriguez, C., Seale, D., Sherman, M., Smith-Jones, M., Tago, D., Tokmakov, P., Tran, M., Hoorick, B. V., Vasiljevic, I., Zakharov, S., Zolotas, M., Ambrus, R., Fetzer-Borelli, K., Burchfiel, B., Kress-Gazit, H., Feng, S., Ford, S., and Tedrake, R. (2025b). A careful examination of large behavior models for multitask dexterous manipulation.

Thomason, J., Shridhar, M., Bisk, Y., Paxton, C., and Zettlemoyer, L. (2022). Language grounding with 3d objects. In *Conference on Robot Learning*, pages 1691–1701. PMLR.

Tomasello, M., Savage-Rumbaugh, S., and Kruger, A. C. (1993). Imitative learning of actions on objects by children, chimpanzees, and enculturated chimpanzees. *Child development*, 64(6):1688–1705.

Torabi, F., Warnell, G., and Stone, P. (2018). Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*.

Tschernezki, V., Laina, I., Larlus, D., and Vedaldi, A. (2022). Neural feature fusion fields: 3d distillation of self-supervised 2d image representations. *arXiv preprint arXiv:2209.03494*, pages 443–453.

Uppal, S., Agarwal, A., Xiong, H., Shaw, K., and Pathak, D. (2024). Spin: Simultaneous perception, interaction and navigation.

Urakami, Y., Hodgkinson, A., Carlin, C., Leu, R., Rigazio, L., and Abbeel, P. (2019). Doorgym: A scalable door opening environment and baseline agent. *CoRR*, abs/1908.01887.

Van Den Oord, A., Vinyals, O., et al. (2017). Neural discrete representation learning. *Advances in neural information processing systems*, 30.

Vasuki, A. and Vanathi, P. (2006). A review of vector quantization techniques. *IEEE Potentials*, 25(4):39–47.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, volume 30, pages 5998–6008.

Veeriah, V., Oh, J., and Singh, S. (2018). Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*.

Virgolino Soares, J. C., Medeiros, V. S., Abati, G. F., Becker, M., Caurin, G., Gattass, M., and Meggiolaro, M. A. (2023). Visual localization and mapping in dynamic and changing environments. *Journal of Intelligent & Robotic Systems*, 109(4):95.

Vora, S., Radwan, N., Greff, K., Meyer, H., Genova, K., Sajjadi, M. S., Pot, E., Tagliasacchi, A., and Duckworth, D. (2021). Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *arXiv preprint arXiv:2111.13260*.

WAGO Kontakttechnik GmbH & Co. KG (2025). Wago — electrical interconnection and automation technology. Retrieved May 24, 2025, from https://www.wago.com.

Wake, N., Kanehira, A., Sasabuchi, K., Takamatsu, J., and Ikeuchi, K. (2023). Gpt-4v(ision) for robotics: Multimodal task planning from human demonstration. *arXiv preprint arXiv:2311.12015*.

Walke, H., Black, K., Lee, A., Kim, M. J., Du, M., Zheng, C., Zhao, T., Hansen-Estruch, P., Vuong, Q., He, A., Myers, V., Fang, K., Finn, C., and Levine, S. (2023). Bridgedata v2: A dataset for robot learning at scale.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023a). Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*.

Wang, H., Wang, W., Liang, W., Xiong, C., and Shen, J. (2021). Structured scene memory for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8455–8464.

Wang, J. M., Fleet, D. J., and Hertzmann, A. (2007). Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298.

Wang, Y., Chao, W.-L., Weinberger, K. Q., and van der Maaten, L. (2019). Simpleshot: Revisiting nearest-neighbor classification for few-shot learning.

Wang, Y., Li, Z., Zhang, M., Driggs-Campbell, K., Wu, J., Fei-Fei, L., and Li, Y. (2023b). D3 fields: Dynamic 3d descriptor fields for zero-shot generalizable robotic manipulation. *arXiv preprint arXiv:2309.16118*.

Wei, B., Ren, M., Zeng, W., Liang, M., Yang, B., and Urtasun, R. (2021). Perceive, attend, and drive: Learning spatial attention for safe self-driving. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4875–4881. IEEE.

Werby, A., Huang, C., Büchner, M., Valada, A., and Burgard, W. (2024). Hierarchical

open-vocabulary 3d scene graphs for language-grounded robot navigation. *arXiv preprint arXiv:2403.17846*.

Wong, J., Tung, A., Kurenkov, A., Mandlekar, A., Fei-Fei, L., Savarese, S., and Martín-Martín, R. (2022). Error-aware imitation learning from teleoperation data for mobile manipulation. In *Conference on Robot Learning*, pages 1367–1378. PMLR.

Wu, C., Huang, L., Zhang, Q., Li, B., Ji, L., Yang, F., Sapiro, G., and Duan, N. (2021). Godiva: Generating open-domain videos from natural descriptions. *arXiv preprint arXiv:2104.14806*.

Wu, J., Chong, W., Holmberg, R., Prasad, A., Gao, Y., Khatib, O., Song, S., Rusinkiewicz, S., and Bohg, J. (2024). Tidybot++: An open-source holonomic mobile manipulator for robot learning. In *Conference on Robot Learning*.

Wu, P., Shentu, Y., Yi, Z., Lin, X., and Abbeel, P. (2023). Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. *arXiv preprint arXiv:2309.13037*.

Wu, Y., Tucker, G., and Nachum, O. (2019). Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*.

Wu, Z., Xiong, Y., Yu, S., and Lin, D. (2018). Unsupervised feature learning via non-parametric instance-level discrimination.

Wulfmeier, M., Ondruska, P., and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

Xiao, T., Radosavovic, I., Darrell, T., and Malik, J. (2022). Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*.

Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., and Sridhar, S. (2022). Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library.

Yadav, K., Ramrakhya, R., Ramakrishnan, S. K., Gervet, T., Turner, J., Gokaslan, A., Maestre, N., Chang, A. X., Batra, D., Savva, M., et al. (2023). Habitat-matterport 3d semantics dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4927–4936.

Yan, C., Qu, D., Xu, D., Zhao, B., Wang, Z., Wang, D., and Li, X. (2024a). Gs-slam: Dense visual slam with 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19595–19604.

Yan, Z., Li, S., Wang, Z., Wu, L., Wang, H., Zhu, J., Chen, L., and Liu, J. (2024b). Dynamic open-vocabulary 3d scene graphs for long-term language-guided mobile manipulation. *arXiv preprint arXiv:2410.11989*.

Yang, J., Chen, X., Qian, S., Madaan, N., Iyengar, M., Fouhey, D. F., and Chai, J. (2023). Llm-grounder: Open-vocabulary 3d visual grounding with large language model as an agent.

Yang, J., Glossop, C., Bhorkar, A., Shah, D., Vuong, Q., Finn, C., Sadigh, D., and Levine, S. (2024a). Pushing the limits of cross-embodiment learning for manipulation and navigation.

Yang, R., Lu, Y., Li, W., Sun, H., Fang, M., Du, Y., Li, X., Han, L., and Zhang, C. (2022). Rethinking goal-conditioned supervised learning and its connection to offline rl. *Iclr*.

Yang, S., Liu, M., Qin, Y., Ding, R., Li, J., Cheng, X., Yang, R., Yi, S., and Wang, X. (2024b). Ace: A cross-platform visual-exoskeletons system for low-cost dexterous teleoperation.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021a). Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021b). Reinforcement learning with prototypical representations. *arXiv preprint arXiv:2102.11271*.

Yenamandra, S., Ramachandran, A., Khanna, M., Yadav, K., Chaplot, D. S., Chhablani, G., Clegg, A., Gervet, T., Jain, V., Partsey, R., Ramrakhya, R., Szot, A., Yang, T.-Y., Edsinger, A., Kemp, C., Shah, B., Kira, Z., Batra, D., Mottaghi, R., Bisk, Y., and Paxton, C. (2023a). The homerobot open vocab mobile manipulation challenge. In *Thirty-seventh Conference on Neural Information Processing Systems: Competition Track*.

Yenamandra, S., Ramachandran, A., Yadav, K., Wang, A., Khanna, M., Gervet, T., Yang, T.-Y., Jain, V., Clegg, A. W., Turner, J., et al. (2023b). Homerobot: Open-vocabulary mobile manipulation. *arXiv preprint arXiv:2306.11565*.

Yokoyama, N., Clegg, A., Truong, J., Undersander, E., Yang, T.-Y., Arnaud, S., Ha, S., Batra, D., and Rai, A. (2023). ASC: Adaptive skill coordination for robotic mobile manipulation. *arXiv preprint arXiv:2304.00410*.

Yokoyama, N., Ha, S., and Batra, D. (2021). Success weighted by completion time: A dynamics-aware evaluation criteria for embodied navigation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1562–1569.

Yokoyama, N., Ha, S., Batra, D., Wang, J., and Bucher, B. (2024). Vlfm: Vision-language frontier maps for zero-shot semantic navigation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 42–48. IEEE.

Young, S., Gandhi, D., Tulsiani, S., Gupta, A., Abbeel, P., and Pinto, L. (2020). Visual imitation made easy. *arXiv e-prints*, pages arXiv–2008.

Young, S., Pari, J., Abbeel, P., and Pinto, L. (2021). Playful interactions for representation learning. *arXiv preprint arXiv:2107.09046*.

Yu, C., Liu, Z., Liu, X.-J., Xie, F., Yang, Y., Wei, Q., and Fei, Q. (2018). Ds-slam: A semantic visual slam towards dynamic environments. In *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1168–1174. IEEE.

Yu, K.-T., Bauza, M., Fazeli, N., and Rodriguez, A. (2016). More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 30–37. IEEE.

Yu, P., Guo, C., Liu, y., and Zhang, H. (2021). Fusing semantic segmentation and object detection for visual slam in dynamic scenes. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, pages 1–7.

Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., and Tagliasacchi, M. (2021). Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507.

Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., et al. (2022). Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*.

Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhwani, V., et al. (2020). Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*.

Zeng, K.-H., Zhang, Z., Ehsani, K., Hendrix, R., Salvador, J., Herrasti, A., Girshick, R., Kembhavi, A., and Weihs, L. (2024). Poliformer: Scaling on-policy rl with transformers results in masterful navigators. *arXiv preprint arXiv:2406.20083*.

Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., and Urtasun, R. (2019). End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8660–8669.

Zhai, X., Mustafa, B., Kolesnikov, A., and Beyer, L. (2023). Sigmoid loss for language image pre-training.

Zhan, A., Zhao, P., Pinto, L., Abbeel, P., and Laskin, M. (2020). A framework for efficient robotic manipulation. *arXiv preprint arXiv:2012.07975*.

Zhang, C., Meng, X., Qi, D., and Chirikjian, G. S. (2024). Rail: Robot affordance imagination with large language models. *arXiv preprint arXiv:2403.19369*.

Zhang, L., Wei, L., Shen, P., Wei, W., Zhu, G., and Song, J. (2018a). Semantic slam based on object detection and improved octomap. *IEEE Access*, 6:75545–75559.

Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K., and Abbeel, P. (2018b). Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *ICRA*, pages 5628–5635. IEEE.

Zhao, L., Yang, T., Yang, Y., and Yu, P. (2023a). A wearable upper limb exoskeleton for intuitive teleoperation of anthropomorphic manipulators. *Machines*, 11(4):441.

Zhao, R., Sun, X., and Tresp, V. (2019). Maximum entropy-regularized multi-goal reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 7553–7562. PMLR, PMLR.

Zhao, T. Z., Kumar, V., Levine, S., and Finn, C. (2023b). Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea.

Zhao, X., Agrawal, H., Batra, D., and Schwing, A. G. (2021). The surprising effectiveness of visual odometry techniques for embodied pointgoal navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16127–16136.

Zheng, R., Cheng, C.-A., Daumé III, H., Huang, F., and Kolobov, A. (2024). Prise: Llm-style sequence

compression for learning temporal action abstractions in control. In *Forty-first International Conference on Machine Learning*.

Zhi, S., Sucar, E., Mouton, A., Haughton, I., Laidlow, T., and Davison, A. J. (2021). ilabel: Interactive neural scene labelling. *arXiv preprint arXiv:2111.14637*.

Zhou, H., Ding, M., Peng, W., Tomizuka, M., Shao, L., and Gan, C. (2023a). Generalizable long-horizon manipulations with large language models. *arXiv preprint arXiv:2310.02264*.

Zhou, W., Jiang, B., Yang, F., Paxton, C., and Held, D. (2023b). Learning hybrid actor-critic maps for 6d non-prehensile manipulation. *arXiv preprint arXiv:2305.03942*.

Zhou, X., Girdhar, R., Joulin, A., Krähenbühl, P., and Misra, I. (2022). Detecting twenty-thousand classes using image-level supervision. *arXiv preprint arXiv:2201.02605*, pages 350–368.

Zhu, X., Tian, R., Xu, C., Huo, M., Zhan, W., Tomizuka, M., and Ding, M. (2023). Fanuc manipulation: A dataset for learning-based manipulation with fanuc mate 200iD robot. https://sites.google.com/berkeley.edu/fanuc-manipulation.

Zhu, Y., Wang, Z., Merel, J., Rusu, A. A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., and Heess, N. (2018). Reinforcement and imitation learning for diverse visuomotor skills. *CoRR*, abs/1802.09564.

Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., Vuong, Q., Vanhoucke, V., Tran, H., Soricut, R., Singh, A., Singh, J., Sermanet, P., Sanketi, P. R., Salazar, G., Ryoo, M. S., Reymann, K., Rao, K., Pertsch, K., Mordatch, I., Michalewski, H., Lu, Y., Levine, S., Lee, L., Lee, T.-W. E., Leal, I., Kuang, Y., Kalashnikov, D., Julian, R., Joshi, N. J., Irpan, A., brian ichter, Hsu, J., Herzog, A., Hausman, K., Gopalakrishnan, K., Fu, C., Florence, P., Finn, C., Dubey, K. A., Driess, D., Ding, T., Choromanski, K. M., Chen, X., Chebotar, Y., Carbajal, J., Brown, N., Brohan, A., Arenas, M. G., and Han, K. (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*.

Ziv, A., Gat, I., Lan, G. L., Remez, T., Kreuk, F., Défossez, A., Copet, J., Synnaeve, G., and Adi, Y. (2024). Masked audio generation using a single non-autoregressive transformer. *arXiv preprint arXiv:2401.04577*.