Building Blocks for Human-AI Alignment: Specify, Inspect, Model, and Revise

by

Serena Lynn Booth

A.B., Harvard University (2016)
S.M., Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by: Serena Lynn Booth
Department of Electrical Engineering and Computer Science
October 9, 2023

Certified by: Julie A. Shah
H.N. Slater Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Building Blocks of Human-AI Alignment: Specify, Inspect, Model, and Revise

by

Serena Lynn Booth

Submitted to the Department of Electrical Engineering and Computer Science
on October 9, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The learned behaviors of AI systems and robots should align with the intentions of their human designers. In service of this goal, people—*especially* experts—must be able to easily specify, inspect, model, and revise AI system and robot behaviors. These four interactions are critical building blocks for human-AI alignment. In this thesis, I study each of these problems. First, I study how experts write reward function specifications for reinforcement learning (RL). I find that these specifications are written with respect to the RL algorithm, not independently, and I find that experts often write erroneous specifications that fail to encode their true intent, even in a trivial setting [22]. Second, I study how to support people in inspecting the agent's learned behaviors. To do so, I introduce two related Bayesian inference methods to find examples or environments which invoke particular system behaviors; viewing these examples and environments is helpful for conceptual model formation and for system debugging [25, 213]. Third, I study cognitive science theories that govern how people build conceptual models to explain these observed examples of agent behaviors. While I find that some foundations of these theories are employed in typical interventions to support humans in learning about agent behaviors, I also find there is significant room to build better curricula for interaction—for example, by showing counterexamples of alternative behaviors [24]. I conclude by speculating about how these building blocks of human-AI interaction can be combined to enable people to revise their specifications, and, in doing so, create better aligned agents.

Thesis Supervisor: Julie Shah
Title: H.N. Slater Professor of Aeronautics and Astronautics

To my teachers, most of all my mother

I first thank Julie Shah, my advisor. When you tell someone Julie is your advisor they say something to the effect of: "oh, how nice!" And this is true: Julie's kind optimism is ever-reassuring. Perhaps the thing I am most grateful for over these years was Julie's invitation to contribute to MIT's Social and Ethical Responsibilities in Computing efforts through many different avenues. Though none of these contributions are included in this thesis, I believe it to be my most impactful work thus far.

I am also extremely grateful to my committee. Elena Glassman, thank you for treating me as a peer, even when I was just a wee second year with unbridled confidence. I will always remember the respect you offered me, and I will attempt to model this in my own mentoring. Dylan Hadfield-Menell, thank you for encouraging me to work in reward design, and for cheering me on over the years with thoughtful support and hallway chats. Peter Stone, thank you for providing thoughtful and detailed advice, and for inviting me to be an extra in your group. Leslie Kaelbling, thank you for giving me the opportunity to design materials for your machine learning class and for providing advice at pivotal moments of stress during the Ph.D.

I repeatedly won the collaborator lottery. Brad Knox, thank you for inspiring me to work on reward design and for your extensive support as I first ventured into this subdomain. Yilun Zhou, it is hard to imagine how my Ph.D. would have gone without your thoughtful input and our exciting collaborations. Nadia Figueroa, thank you for sharing your incredible competency with me. Ankit Shah, thank you for your endless patience teaching me when I knew nothing. Aspen Hopkins, thank you for teaching me to conduct quality qualitative research. Marco Ribeiro, thank you for sharing your critical approach to research. Scott Niekum, thank you for bringing your smart, critical, and supportive approach to our collaborations. Christian Muise, thank you for being invested in my growth when I was just a first year student, and for supporting me from afar with boundless enthusiasm over the years.

This dissertation is dedicated to my teachers. I am indebted to Radhika Nagpal, Krzystof Gajos, Jim Waldo, Harry Lewis, L.G. Isom, Suzanne Wexler, Angel Perkins, et al. I am also grateful to my students: to those who have taken my classes, to those who have been patient as I made mistakes and learned with you, and especially to the undergrad researchers who have worked with me: Tiffany Horter, Yiming Zheng, Reagan Zimmerman, Emily Levenson, Rene Reyes, Paul Calvetti, and Melissa Calvert.

Moving 3,000 miles away from my friends Juan Posadas and Chris van Harmelen to start this endeavor hurt like hell. Juan and CVH made a bet that I would not finish. Juan bet against: he claims he imagined I would quit and pursue some more lucrative adventure. Juan was, of course, wrong, as he seemingly always is, and now owes Chris *yet another* beer. A surprising upside to Juan and I both leaving San Francisco was that CVH managed to thrive without our constant, unsolicited, and objectively terrible dating advice, and is—gasp!—a married man now. I love you both, and I will continue to dream of a future where we all live in the same city.

When I started my Ph.D., my friend Sharon Zhou was pursuing her PhD 3,000 miles away. Our phone calls and complaints and shared frustrations made everything easier, and I have been fortunate to have a front row seat watching Sharon chase her dreams after graduating. My friend Daniel Citron, I fondly remember watching Fleabag together on my sofa and becoming increasingly silly people as we were indoctrinated. I am also thankful for my long-time family friends, Béla and Gabriella Bollobás and Jim Blythe and Sheila Martin; the many parties and dinners at your homes during my childhood and adolescence shaped my spirit. Shoutouts, also, to Monica Garde, Alex Bondarenko, and Hendrik Strobelt.

At MIT, I have made many friends and been inspired by many people: Lily Tsai, Kru Kikkeri, Monica Agrawal, Divya Shanmugam, Sarah Cen, Aspen Hopkins, Irene Chen, Willie Boag, Josh Robinson, Jonas Lehmann, Alexandra (Ola) Zytek, Bianca Lepe, Botond Oreg, Bazyli Szymański, Karima Ma, Tan Zhi Xuan, Alex Lew, Natasha Seelam, Camille Biscarrat, Leilani Gilpin, James Salamy, Hannah Varner, Andreas Haupt, Andi Bauer, Sarah Muschinske, Stella Lau, Rachel Holladay, Harini Suresh, Isaac Harris, Jodie Miu, Hannah Varner, Jules Nazaré, Milo Phillips-Brown, Agnes Villanyi, Erjona Topalli, Melissa Landman, Genevieve Flaspohler, Marion Boulicault, Kate Stoll, Anna Zeng, and Peter Satterthwaite. I am very sorry for the many I am forgetting. May our community continue to prosper as we disperse across the world!

I am grateful for my organizations at MIT and for the people who compose these organizations: the Science Policy Initiative, Grad Women of Course 6, the MIT European Club, and the Interactive Robotics Group. In IRG, I am particularly grateful for Yilun Zhou, Chris Fourie, Mycal Tucker, Ankit Shah, Joseph Kim, and Pem Lasota.

My sister, Lara, is my best friend, lifelong playmate, and, wonderfully, also my peer as an MIT EECS Ph.D. student. I owe a lot in life to my brother, Jo; I would certainly not have made it to Harvard without Jo's influence. As children I tried relentlessly to compete with him—almost always coming up short, in part because Jo is two years my senior, and in part because Jo is shockingly clever. My sister Alex, thank you for your influence in pushing me toward studying computer science and working at Google. My brother Jason, I am always rooting for your success.

My mother embodies a love of learning more than anyone else I have ever met; extremely stiff competition! She has taught me more than anyone else ever could, and she devoted so much of her life to growing me and my four siblings into smart, successful, and capable people. There is not a chance I would have pursued this endeavor without her love, care, and support. Thank you, Mummy.

I must confess: I played the Ph.D. on easy. I used the illicit JHT cheatcode! But this isn't just a cheatcode for the Ph.D.; he is my cheatcode for (my) life. I love you.

Lastly, my research requires user studies. I am extremely grateful to the many participants who have spent their time contributing to this science over the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The behaviors of AI systems, including embodied AI systems like robots, should align with human intents and values [38]. However, exactly what alignment means and how it should be established and measured is a highly contested topic in the AI research community [29]. This dissertation reasons about alignment in terms of key interactions. When writing a specification for AI, the human should be able to construct a reasonable belief over how the specification will manifest as a behavioral policy: the probability of the agent taking some action in a given state. Once the AI system has subsequently learned a policy from the specification, the human should be able to generally predict how the AI system will act in response to encountered states, both novel and known. To achieve this, the human must have sufficient insight into the AI system's decision-making mechanism to build a useful conceptual model of its knowledge, capabilities, and limitations—and such insight should allow the human to determine when and when not to appropriately use and rely on the AI system.

This dissertation focuses on the building blocks necessary to support humans throughout these alignment-critical interactions: helping humans specify behaviors, inspect learned behaviors, form conceptual models, and, ultimately, revise their specifications. How can we better support people in these joint tasks of writing specifications for AI systems and interpreting these systems' learned behaviors?

Specifying Behaviors Reinforcement learning (RL) is a promising approach for building AI systems. Reward functions are an exceptionally flexible framework for specifying behaviors, and, as such, there is tremendous optimism about RL's potential; some researchers argue that reward functions can specify the nature of intelligence [182]. Nonetheless, RL's usefulness is significantly impaired by the difficulty of specifying the reward function, which can be misspecified or underspecified [7]. This dissertation contributes a study of the human reward design process [22]. First, this dissertation shows that reward functions can easily be overfit to learning algorithms, wherein the reward function is overloaded to both encode the desired behavior and also facilitate fast and successful learning for a specific algorithm or hyperparameter choice, at the expense of interoperability and generality. Second, this dissertation includes a user study to assess whether this problem of overfitting equally manifests with human experts designing the reward functions. While this dissertation confirms this problem of overfitting is indeed persistent, this dissertation also discovers that—in a trivial gridworld environment—the majority of expert humans write incorrect reward functions. This dissertation attributes these failures to the mismatched interpretations of the reward function between the human designers and the goals of RL algorithms writ large. Humans view reward functions with a myopic lens, as a mechanism for encoding the relative goodness of each possible state, but the typical RL objective is instead to maximize the cumulative discounted return. This first study raises the questions: how can we enable humans to write better reward functions, and how can we enable robots to better interpret flawed reward functions?

Inspecting Behaviors After learning a behavioral policy from a specification, how can a person assess whether the AI has learned behaviors that meets their expectations (i.e., is aligned to their intent)? The most common practice is to observe examples of the AI acting in a randomly-selected or hand-curated set of environments. Without adding structure and discipline to this practice, this observation process is limited in its usefulness. Instead, this dissertation proposes supporting humans in searching for examples that communicate specific and targeted behaviors. To this

end, this dissertation first introduces a method for inspecting the behaviors of neural network or other classifiers. In BAYES-TREX [25], a user specifies a prediction target and a generative model, and uses Bayesian inference to find examples that meet the prediction target. For instance, the target might be to find examples that are scored ambiguously across two classes. BAYES-TREX helps with debugging and understanding neural networks, as it can be used to find *ambiguous* examples to communicate class boundaries or highly-confident incorrect classifications to communicate systematic failures. This dissertation subsequently adapts this approach to create ROCUS, a method for debugging and improving robot controller behaviors by finding environments in which interesting behavior occurs [213]. Moreover, this dissertation finds ROCUS to be helpful for the successful design of a dynamical system-based robot controller. ROCUS can also be used to assess the behaviors learned through RL with a user-designed reward function and can thus be applied to help the human methodically iterate on the design of a reward function specification.

Building Conceptual Models How do humans come to understand the behavioral patterns encoded in a reward function, or learned by an AI system through this reward function? More generally, how do humans maintain and mitigate uncertainty about their own beliefs about AI systems' capabilities and limitations? This uncertainty relates to the human ability to form conceptual models, which are abstract models used for reasoning. The storied study of human concept learning from the learning sciences [135, 62] provides a rough blueprint for how to help people build and update accurate and flexible conceptual models and can be leveraged for human-AI interaction (HAI). These human concept learning theories assert that conceptual models are best formed by experiencing examples that follow highly-structured patterns of variance and invariance [135], and by experiencing structurally-aligned analogous examples [62], which support rapid knowledge transfer. When interacting with an AI system, a person will inevitably develop a conceptual model of the system's behaviors. But, without structure to their learning, the resulting conceptual model may be incorrect or inflexible. This dissertation studies how these theories of human concept

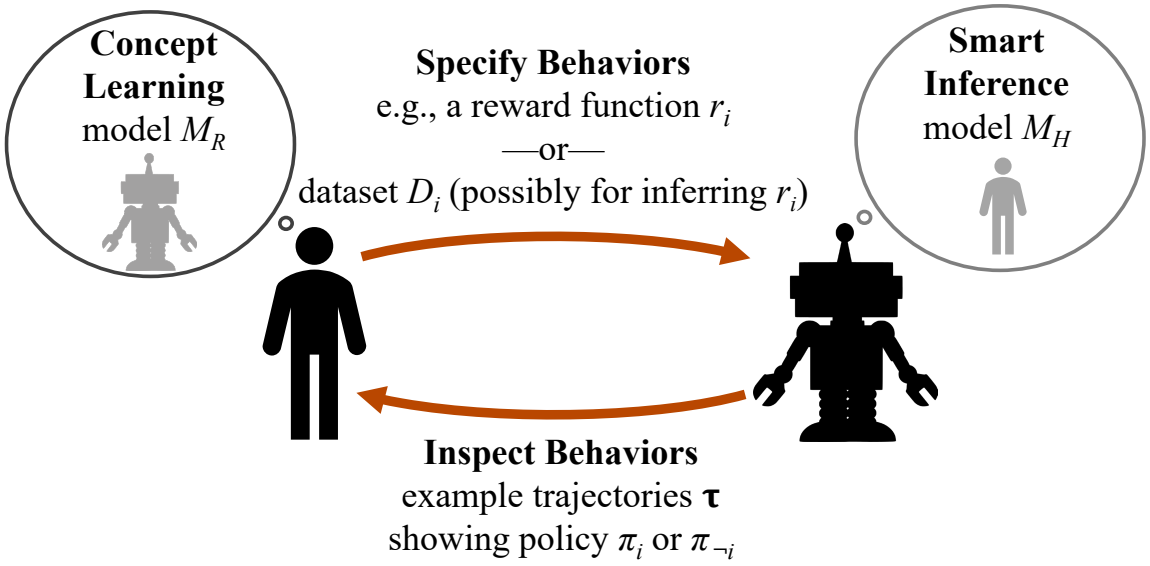


Figure 1-1: The vision of this dissertation for aligning AI system behaviors with human intents, using a robot as an example: A human provides a specification either in the form of a candidate reward function r_i or some other signal (e.g., a dataset of preferences D_i [39, 106]) that encodes their desired behaviors for the robot. Then, the robot models the human (M_H) and updates its behavioral policy (π_i). The robot presents strategically-selected examples of its learned behavior to the human, perhaps in the form of example trajectories, environments, or some form of explanation. The robot may also present counterexamples to support the human's learning (i.e., drawn from a counterexample policy, $\pi_{\neg i}$). In turn, the human updates their model of the robot (M_R) that represents their understanding of the robot's knowledge, capabilities, and limitations. This process is iterative: the human may update their specification and repeat these interactions until they believe the system is sufficiently well aligned. This dissertation studies how we can support each of these interactions.

learning should be adapted for human-AI interaction (HAI): the analysis of 35 HAI works shows ad-hoc incorporation of *some* of these patterns [24], but that the research community still has many blind spots. For example, it is still exceedingly rare to show counterexamples of capabilities, even though these learning theories demonstrate that counterexamples are essential for establishing the bounds of capabilities. This dissertation further provides design guidance for better structuring human observations of AI systems' learned behaviors in service of conceptual model formation.

Combining These Building Blocks This dissertation studies three building blocks for human-AI alignment: how to specify, inspect, and model the behaviors of AI systems. Combining these building blocks can inform the design of an iterative, interactive, and interleaved learning system, as presented in Figure 1-1. An AI system designed in this manner should be better aligned to the interests and values of the human designer, and the human designer should better understand the capabilities and limitations of the system such that they are better positioned to determine when and when not to rely on the AI system; the ultimate goal of alignment!

1.1 Specifying Behaviors: Chapter 3

Chapter 3 studies how experts write reward functions as specifications for AI systems and is based on Booth et al.'s AAAI 2023 paper, "The Perils of Trial-and-Error Reward Design: Misdesign through Overfitting and Invalid Task Specifications" [22]. This study originated from reasoning about the practitioner's approach to RL, especially writing specifications in the forms of reward functions. While there is a significant research thrust to move toward learned reward functions [39], reward functions are often still hand-crafted by domain experts to great success. For example, the reward functions used to create a super-human agent in the popular video game Gran Turismo were hand-engineered [205]. Though trial-and-error reward design is discouraged in the RL community [190, 177], experts often use it nonetheless [103].

This study of experts' reward design process was borne from these observations and explores two questions in reward design. First, do experts overfit the design of reward functions with respect to their choice of RL algorithm and hyperparameters? Overfitting in this context means that the reward function has been excessively tailored to a particular RL algorithm, which raises concerns about the generality of these reward functions as specifications. Moreover, it also calls into question the correctness of these reward functions when optimized with varying RL algorithms, since the reward is so tailored to a particular optimizer that changing this optimizer could result in unexpected learning outcomes or potentially even failures.

Second, and perhaps of greater importance for this dissertation: are experts generally able to write correct hand-designed reward functions, whether by trial-and-error or any other means? If not, this has significant implications for the design of the iterative system proposed in Figure 1-1. If experts make systematic errors in the design of reward functions, this presents an opportunity to inform the design of helpful interventions to prevent or overcome such errors. If experts make many unsystematic errors, this suggests that the benefits of hand-coded reward functions may not outweigh the costs, and supports a competing line of research that pushes for learned reward functions in place of hand-coded ones. Thus, this inquiry seeks both to uncover the prevalence of incorrect reward function design and also to explore the specific types of mistakes that experts commonly make.

To meaningfully study these questions, this dissertation needs to establish the definition of a "correct" reward function. In some settings, the correct reward function is relatively unambiguous: in a game like Chess, the reward function simply determines whether the agent wins or loses, and we can all more or less agree about the correctness of this reward function. In most settings, however, determining the correct reward function is not possible. This dissertation defines a correct reward function to be an aligned reward function, but alignment is again an amorphous concept that is defined by some combination of intuition and desiderata, so this definition is not pragmatic. To remove this ambiguity, this dissertation provides a groundtruth reward function to experts in the form of natural language for this study on specifying behaviors. This makes the proposed study tractable: the task is not to write a correct reward function with respect to an expert's values; rather, it is to write down a reward function that encodes the given natural language specification.

The first component of this chapter is an extensive computational study, which assesses whether reward functions can exhibit the problem of overfitting in principle. This first component omits the role of a human reward designer. The second component introduces the human and all of the associated complexity and noise. This second component consists of a naturalistic user study to assess whether this problem of overfitting is not just a concern in principle but also in practice. This naturalistic

user study also seeks to uncover common classes of errors these experts make when writing reward functions, which can be used as an input for reasoning about how to resolve any such errors.

These studies both use the same domain for analysis: the Hungry Thirsty environment, first introduced by Singh et al. [184]. This is a seemingly-trivial gridworld environment with moderately complicated dynamics; this environment might appear in an introductory class on reinforcement learning. The objective, stated in natural language, is to "teach an agent to eat as much as possible. There's a catch, though; the agent can only eat when it's not thirsty." A reasonable interpretation of this natural language specification is that the agent achieves some reward $r > 0$ for successfully eating, and 0 otherwise. While it is tempting to reward the agent for successfully drinking, it is not necessary. Under this natural language specification, the optimal policy for an agent is to alternate between traversing to the grid cell that contains the water and drinking when the agent is thirsty and traversing to the grid cell that contains the food and eating when not thirsty. Because of the discrete nature of the environment, a large class of reward functions encode the optimal policy—including reward functions that are *not* shaped with a potential function. This property of admitting many correct reward functions makes Hungry Thirsty a compelling domain to study the reward design process, both computationally and with human experts.

The computational study explores whether different reward functions achieve more or less rapid success in approximating the optimal policy derived from the natural language specification when varying the RL algorithm and hyperparameter choices. Specifically, for each choice of RL algorithm and hyperparameter selection, this dissertation trains 10 agents for each of 5,196 different candidate reward functions. This dissertation scores the performance of these trained agents against the given reasonable interpretation of the natural language specification. Across these many experiments, this dissertation finds consistent and strong evidence of overfitting. This dissertation first finds that the best-performing reward functions are different with the varying choices of algorithms and hyperparameters. Moreover, this dissertation finds that the performance of reward functions is uncorrelated across different choices of algorithm

and hyperparameters. This latter finding particularly indicates that this problem of overfitting is not an obscure phenomenon, but a common and even expected phenomenon, at least in the studied Hungry Thirsty domain. From these experiments, this dissertation concludes that overfitting is a concern, at least in principle.

The naturalistic user study explores how experts design reward functions in practice. 30 experts use a Jupyter code notebook to train RL agents to solve the Hungry Thirsty domain, and they have flexibility in choosing the reward function as well as the algorithm and hyperparameters. The order in which they are prompted to fill in these details is randomized. Similar to the computational setting, this disseration finds evidence that experts also overfit their reward functions to their choice of algorithm and hyperparameters. Additionally, this dissertation finds that many experts (53%) design reward functions that do not encode the natural language specification, despite this domain being largely trivial. This latter finding encourages two interventions. First, this finding supports the endeavor to learn reward functions instead of hand-coding them; however, we must also assess whether learned reward functions successfully encode the desired specification! Second, this finding can be used to reason about how to improve the hand-coded reward design process. By analyzing experts' thought processes, this dissertation finds that temporal discounting is particularly difficult for people to reason about, so future interventions can be designed to assist in this specific manner. Additionally, experts in the user study observed the agent acting in random environment instantiations—but more systematic analysis, for example using the techniques discussed in Chapters 5 and 6 can more readily expose the errors in specifications.

Overall, this chapter on specifying behaviors explores the challenges of hand-designing reward functions and using reward functions as specifications more generally. Hand-designed reward functions remain desirable, in part because a human engineer has control and insight into the expected behaviors of the system. This chapter contributes evidence both of overfitting and of misspecification when experts hand-design reward functions. While there is cause for pessimism, these findings also offer insights into how to retool the reward design process to help experts craft better

specifications, which is cause for optimism. In particular, Chapters 4 and 5 introduce a mechanism for systematically inspecting an agent's behaviors—behaviors that may be the consequence of learning a policy with RL using a hand-designed reward function. Chapter 6 builds on these two contributions by discussing *how* experts should use the tooling from Chapter 5 to diagnose and, hopefully, fix misspecifications.

1.2 Inspecting Behaviors: Chapters 4 and 5

Chapters 4 and 5 study how humans can systematically inspect the behaviors of classifiers and robot controllers, respectively. Chapter 4 is based on Booth et al.'s AAAI 2021 paper, "Bayes-TrEx: A Bayesian Sampling Approach to Model Transparency by Example" [25], and Chapter 5 is based on Zhou et al.'s CoRL 2021 paper, "RoCUS: Robot Controller Understanding via Sampling" [213]. These contributions stem from initially thinking about how neural networks make decisions, and how these decisions differ from human decisions. The original conception came from thinking about how, in the quest for alignment in classification settings, neural networks should perhaps make the same errors as a human and was inspired, in part, by Scenic [56]. However, no part of the training procedure directly reinforces this outcome. Thus we imagined designing a method that would uncover ambiguous examples, where the neural network was uncertain of the correct label. By finding and exposing ambiguous examples to a human, and asking the person to relabel, we believed the neural network could be trained to more closely approximate human-like decisions.

In the process of exploring this question of exposing ambiguous examples for re-labeling, this dissertation established that ambiguity is, of course, only one frontier for alignment in this classification setting. Ambiguous examples are often quite unlikely in a neural network, since neural networks often exhibit the problem of overconfidence [71]. For alignment, resolving overconfident misclassifications is equally if not more important than resolving ambiguities. This dissertation thus contributes a method that allows the human to specify the prediction target—for example, ambiguous between two classes or high confidence in a particular class—and the method

finds examples that match this given target. This first neural network-focused work is presented in Chapter 4; the main contribution is a Bayesian inference method for finding examples for which a classifier gives a user-specified prediction confidence. Chapter 5 adapts this method to the setting of studying robot controllers, including those that are learned by using an RL algorithm to optimize a policy based on a given reward function, which is generally the specification method of choice, as discussed in Chapters 2, 3, and 6.

The rationale behind this adaptation for robot controllers is that, when designing a controller—whether through RL or other means—the designer typically would use one of two choices to evaluate their controller. Most likely, they simply watch the controller act in random environments, and from this, they try to diagnose any issues. As discussed in Chapter 6, this approach is subject to highly fragile and inaccurate conceptual model formation. Alternatively, the designer might construct a suite of tests—undoubtedly a more sensible and systematic approach to testing and conceptual model formation. However, this approach is driven by the environment and not the outcome of executing the robot controller in that environment. Hence, conceptual model formation is still challenging. To address this, the method systematically searches for environments that maximize a human designer's given behavioral metric—for example, in the Hungry Thirsty setting of Chapter 3, the designer might search for environments in which the agent drinks the most water, and, in doing so, uncover the common flaw in their reward function that results in a suboptimal policy when the water in the environment is located particularly far from the food.

When developing RoCUS to inspect the behaviors of robot controllers, this study found it to be directly useful for refining the design of these controllers. Specifically, RoCUS helped inform the design of a dynamical system-based controller for a 7 Degree-of-Freedom (7DoF) robot arm, where the objective was to reach a specific location in the environment. This objective is slightly complicated because reaching the target requires reaching around an obstruction, a table in the way. This study used RoCUS to find environments that led to the robot reaching the target successfully by bringing its end effector close to the goal. When analyzing these successes, a

troubling pattern emerged: all of the environments that led to successful outcomes with minimal end effector distances had the target located in the same half of the environment. By reasoning about this result, the bug in the controller's specification can be identified: the end effector needed additional buffer to prevent it from coming too close to the obstructions. While this controller improvement concerns a dynamical system controller, the same principles carry over to RL and reward function design. Debugging and revising specifications is made easier by systematically analyzing a system's learned behaviors.

Amusingly, a similar method for inspecting the behaviors of robot controllers was developed and published simultaneously at another laboratory [54]. While Chapter 5 uses a Bayesian inference framing for finding environments, Fontaine et al.'s work [54] instead uses a Quality Diversity sampling approach. Functionally, both works are similar: they are both framed as methods for finding environments that improve humans' conceptual model formation and similarly both require user-provided behavioral metrics to do so. This duplicity on the research record should be interpreted as indicating the importance of finding and exposing the right environments to the system designers, as doing so is necessary both for the designer's conceptual model development and also ultimately for attaining the amorphous alignment.

1.3 Building Conceptual Models: Chapter 6

Chapter 6 studies how people who interact with AI systems and robots can and should build conceptual models of these systems' capabilities and limitations. This chapter is based on Booth et al.'s HRI 2022 paper, "Revisiting Human-Robot Teaching and Learning Through the Lens of Human Concept Learning" [24] and Horter et al.'s HRI 2023 workshop paper, "Varying How We Teach: Adding Contrast Helps Humans Learn about Robot Motions" [83]. This chapter of the dissertation is focused on the cognitive processes in play when writing a behavioral specification or interacting with an AI system more generally. This line of work has a slightly different original focus from the other chapters of this dissertation; it was inspired by work on learning

from human feedback, which pointed out that feedback tends to be both strategy-dependent [129, 130] and policy-dependent [133]. These works call into question whether people are easily able to understand a given policy and encourages the design of interventions to support people in developing this understanding.

This inquiry led to the exploration of cognitive theories of human concept learning. This dissertation sought to inform the design of better interactive systems that reinforce people's abilities to form conceptual models of policies, in the dual tasks of teaching an agent (i.e., with feedback) and learning from an agent (i.e., forming a conceptual model of the agent's policy). This dissertation particularly identifies two theories of learning that have been widely and successfully used in classroom settings across varied learning domains—in disciplines ranging from mathematics to language. These two theories are the Variation Theory of Learning [135], which hinges on the idea that experiencing variation of both superficial and critical details is the key to conceptual model formation, and Analogical Transfer Theory [62], which argues that learning is primarily achieved through analogy from familiar contexts to novel ones. With these complementary theories in mind, this dissertation contributes a meta-study that reviews 35 papers from the human-AI interaction (HAI) literature. In this review, this dissertation assesses which principles of these concept learning theories the HAI research record engaged, if any, and how these principles improved the core interactions of alignment. This analysis uncovers some best and worst practices in the existing literature and provides a roadmap for future development.

While this research sought to help people understand policies so that they could give higher quality policy-dependent feedback, it is also instrumental for building the more general conceptual models needed to achieve the interactions described in Figure 1-1, and for reward design as described in Chapter 3. After designing a candidate reward function, the human should assess the learned behaviors of the system—which reflect the combination of the reward function specification and the optimization of this specification. To assess these learned behaviors, the human will look at exemplar behaviors, which can be uncovered with the techniques described in Chapters 4 and 5. But which exemplar behaviors should the human assess? These theories of

concept learning provide guidance to answer this question, by providing insights into the selection, sequence, and presentation of AI system behaviors. The most basic insight in this line of study shows that people experience a significant cognitive benefit from experiencing counterexamples or foils, but that providing such counterexamples is exceedingly rare in the existing literature. This same approach of using counterexamples for concept development has been prescribed by other researchers when thinking about how explanations of AI system behaviors should be designed [139].

1.4 Unanswered Questions & Unexplored Directions

Many unanswered questions and unexplored directions remain. This dissertation assumes an expert is engineering a hand-designed reward function. While this is common (e.g., [205]), an increasing trend in RL assumes, instead, a reward function learned from human preferences [39]. Learned reward functions equally require study to assess their expressivity and humans' propensities for correct and consistent preferences; while my colleagues and I have some preliminary work in this direction [105, 106], this is a ripe direction for future exploration. For both hand-designed and learned reward functions, mechanisms can be designed to compensate for errors, particularly when these errors are systematic; such a future research endeavor can build on the framework of Inverse Reward Design [72]. When exploring the topic of inspecting behaviors, this dissertation makes a significant assumption: that the human is able to specify a behavioral target metric that elucidates the agent's capabilities and limitations. Writing such targets is itself a challenging problem, so future work should explore how to synthesize or guide the human in designing these queries. When exploring how humans build conceptual models for interacting with AI systems and robots, this dissertation primarily relies on data from the existing research literature. Future work should study how each prescribed intervention materially affects human cognition, independently and in combination—and future work should assess this cognition across varying domains, too. These future directions can be read as the outlines of proposed future doctoral theses, and are discussed at length in Chapter 7.

1.5 Other Related Works within my Doctoral Study

Through studying this complex topic of human-AI alignment and the requisite interactions, I published several more loosely related works during my doctoral study.

- *Evaluating the Interpretability of the Knowledge Compilation Map: Communicating Logical Statements Effectively.* Serena Booth, Christian Muise, and Julie Shah. IJCAI 2019 [23].

  An alternative mechanism for inspecting the behaviors of AI systems requires a simplified form of decision logic, such as through logical statements. We study how to best present logical formulas to humans for comprehension, and we find unexpected flexibility in people's ability to reason about logical statements.

- *Do Feature Attribution Methods Correctly Attribute Features?* Yilun Zhou, Serena Booth, Marco Tulio Ribeiro and Julie Shah. AAAI 2022 [214].

  Feature attribution methods are a common approach of inspecting AI system behaviors. Feature attribution methods are designed to identify the salient parts of an input that most contributed to an AI making a given decision. However, most evaluations of feature attribution methods are deficient or incorrect. We provide a protocol for a correct and necessary-but-not-sufficient evaluation of feature attribution methods, and we find feature attribution methods rarely meet the posed necessary condition.

- *Models of human preference for learning reward functions.* W. Bradley Knox, Stephane Hatgis-Kessell, Serena Booth, Scott Niekum, Peter Stone, and Alessandro Allievi. arXiv preprint 2022 [106], and the follow-up to this work: *Learning Optimal Advantage from Preferences and Mistaking it for Reward.* W. Bradley Knox, Stephane Hatgis-Kessell, Sigurdur Orn Adalgeirsson, Serena Booth, Anca Dragan, Peter Stone, and Scott Niekum. arXiv preprint 2023 [105].

  This dissertation repeatedly refers to the dilemma hand-designing or learning a reward function. There are arguments for and against each approach. This contribution discusses how learned reward functions may or may not be able to

identify the correct reward function, and is thus related to Chapter 3. Chapter 7 discusses this work and its implications for future work in more detail.

Due in part to my work studying how to inspect AI system behaviors, I also contributed to an IEEE standard for the transparency of automated decision making.

- *IEEE Standard for Transparency of Autonomous Systems.* Alan Winfield et al., IEEE 2022 [202], and the companion paper that explains the development of this standard: *IEEE P7001: A proposed standard on transparency*, Alan Winfield et al., Frontiers in Robotics and AI 2021 [203].

  This standard defines transparency of autonomous systems as a measurable and testable property. It outlines criteria against which an autonomous system can be scored to provide a numerical assessment of its transparency. These criteria are divided by stakeholders—for example, transparency has different requirements for the general public, for bystanders, and for incident investigators.

Chapter 2

Background

This dissertation draws from and seeks to unify several divergent parts of the research record. Due to this substantive diversity of inputs, there is no single unified background and set of preliminaries to draw from. Instead, the background and preliminaries needed to engage with each chapter are positioned primarily in situ. This chapter formally introduces reinforcement learning, which is used in Chapters 3, 5, and, to some extent, 6. This chapter also briefly discusses a few recurring themes and grounding ideas on the topics of specifications for AI systems and explainable AI.

2.1 Reinforcement Learning (RL)

This thesis, particularly Chapters 3 and 6, assumes basic knowledge of reinforcement learning (RL). In the absence of this prior knowledge, we direct the reader to Sutton and Barto's [190] and Russell and Norvig's [177] excellent introductory texts. In RL, an agent learns a behavioral policy for sequential decision-making based on experience interacting with its environment. An environment can be modeled by a Markov decision process (MDP), which is defined by a tuple $\langle S, A, T, \gamma, D_0, r \rangle$. S and A are the sets of states and actions, respectively. T is a transition function, $T : S \times A \times S \to [0, 1]$. γ is the discount factor and D_0 is the distribution of start states. Lastly, r is a reward function, $r : S \times A \times S \to \mathbb{R}$. An MDP$\backslash\langle \gamma, r \rangle$ is an MDP with neither a discount factor nor a reward function; this formulation is used in Chapter 3 for

studying humans' reward design processes, wherein humans are asked to select the discount factor and reward function. Actions in an MDP can be prescribed by a policy $\pi : S \times A \rightarrow [0, 1]$, where $\tau_\pi = (s_0, a_0, s_1, ...)$ is defined to be a trajectory of states $s_i \in S$ and actions $a_i \in A$ experienced over time by executing π. Discounted return is the discounted sum of reward over a trajectory of length n: $G(\tau) = \sum_{t=0}^{n} \gamma^t r(s_t, a_t, s_{t+1})$. The typical objective in RL is to maximize the expected discounted return, though there are exceptions (e.g., as in distributional RL [44]).

2.2 Writing Specifications

This dissertation focuses on the challenge of writing reward functions as specifications, which is the typical specification formulation in reinforcement learning. Although reward is undeniably a compelling form of specification, with some researchers even arguing that the reward formulation can capture the very nature of intelligence [182], this is by no means the only choice of specification form for AI or robot systems. It is equally possible to write specifications in terms of goals or goal states, e.g., as is typically the case in task and motion planning [59] or dynamical systems modeling approaches [53]. In supervised and unsupervised learning, the specification is typically defined through the choice of loss function; in specific settings, this form of optimization can be applied to sequential decision-making problems characteristic of RL [166]. The choice of specification form and whether it is explicitly defined or learned has implications for a human's ability to construct it correctly [122].

Misspecified Reward Functions In principle, a correct reward function should simply specify "what you want achieved" [190]. At the surface, writing a correct reward function sounds simple, but, in practice, this is exceedingly challenging or even impossible. Reward functions are easily misspecified [7, 103, 161], wherein the reward function contains a "bug" which results in a learning algorithm finding a solution that does not reflect what the reward designer wanted to achieve. Misweighting of a multi-attribute reward function is one common cause of misspecification [161, 22]. A second

common cause of misspecification is the use of proxy rewards; for example, an engineer might provide reward proportional to the average velocity of an autonomous vehicle as a proxy for the average time spent during a commute [161, 103]. Misspecification is nuanced in part because the learning system should have freedom to find creative solutions that may not resemble the solution a human would propose [122], and because reward functions are viewed as a compelling alternative to procedural specification of behaviors. These concerns over the potential misspecification of reward functions, coupled with the advent of extremely flexible learning systems with massive computational power and interconnectivity, has led to increasing concerns over the potential for *catastrophic risk* as more AI systems are developed and deployed [176].

Explicit vs. Learned Reward Functions Reward functions can be explicitly-defined by an expert or learned from an implicit signal, like preferences, feedback, corrections, or demonstrations. Explicitly-defined reward functions are intuitively desirable, since the human is assumed to be able to encode their expert knowledge in this function. Further, the human can be held accountable for the correctness of a hand-designed reward function and the consequences of its deployment—whether rightly or wrongly. However, explicitly-designing a reward function is a non-trivial task [103], so learned reward functions are often favored in practice [39, 159]. However, learned reward functions introduce new surface area for misspecification and so require further study. If the existence of a correct latent reward function is assumed, the learning process should employ inductive biases which are capable of recovering this latent reward function. However, this is not guaranteed under some common reward function learning strategies, even, for example, given the unrealistic assumption of infinite samples of human preferences [106]. Hence, the expressivity of learned reward functions is a compelling direction for future study. This dissertation primarily focuses on reward functions which are explicitly-defined, though the methods and analyses presented in Chapters 5 and 6 equally apply to systems which have learned policies from learned reward functions.

2.3 Inspecting & Explaining Learned Behaviors

Deep learning and deep reinforcement learning methods are often criticized for producing uninterpretable models: it is extremely hard to scrutinize the decision-making processes of these models [157, 128, 45]. In response, there is a large schism in the machine learning community with some researchers calling for the exclusive use of inherently-interpretable models, especially in high-stakes decision-making settings [175]. Such models are ostensibly scrutable by design: a human should be capable of analyzing the reasoning mechanism of the AI model, and can thus at least in principle perfectly understand the knowledge, capabilities, and limitations of the model. In practice, inherently-interpretable models are often equally uninterpretable, either because they require cognitive reasoning beyond what is viable to expect of a person [23] or because they introduce a non-interpretable module or computation as a component of the larger "inherently-interpretable" model (Prototype Networks [124] are one such example). Techniques which aim to explain the decisions of uninterpretable models are also fraught and often even incorrect [1, 214].

In light of these persistent challenges, both in producing inherently interpretable models and in explaining the behaviors of uninterpretable models, we focus instead on simply inspecting the learned behaviors of these models (Chapters 4 and 5), and supporting people in developing conceptual models to explain these behaviors—independently of understanding exactly the process of the underlying decision-making mechanisms (Chapter 6). By supporting people in finding examples which exhibit interesting behaviors and in analyzing these expressed behaviors, this dissertation argues that people can be taught to use these models effectively, even if they do not understand the reasoning mechanisms underlying these models.

Chapter 3

Specify

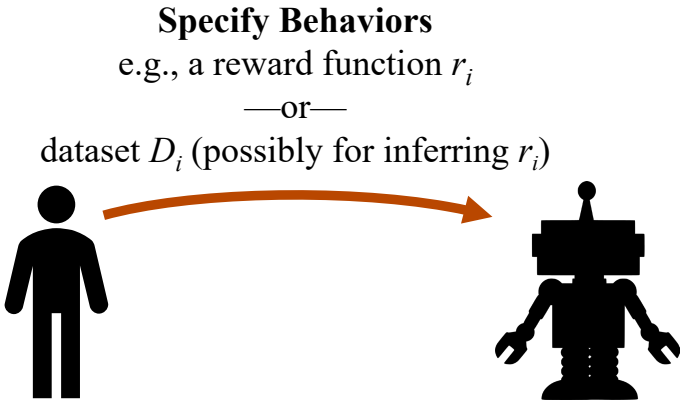


Figure 3-1: This chapter focuses on specifying behaviors for AI systems and robots.

This chapter explores the challenges of designing reward functions as specifications where an expert hand-designs the reward function. This exploration focuses on two questions. First, can reward functions be tailored to the algorithms and hyperparameters which optimize them such that they are overfit? Second, are people able to write "correct" reward functions—and, if not, do they make systematic errors in their reward designs, or are their errors more haphazard? These explorations are focused on the first step required in the interactive system described in Figure 1-1; this focused step is highlighted above in Figure 3-1. The observations uncovered in this chapter inform the design and requirements of the mechanisms for inspecting behaviors (as in Chapters 4 and 5) and for building conceptual models (as in Chapter 6).

3.1 Introduction

In their authoritative text on reinforcement learning, Sutton and Barto ([190]) assert: "The reward signal is your way of communicating to the agent what you want achieved, not how you want it achieved." This statement implies that a reward function should exclusively encode the true task performance metric. Such metrics are often sparse: did the agent succeed at the task or not? Sparse reward functions are rarely used in practice, since it can be hard to learn from sparse signals (for examples, see [208, 9, 103]). As such, the practice of reward design seldom adheres to this adage.[1] Instead, reward functions are typically designed through an ad hoc process of trial and error. In a survey of 24 expert RL practitioners, we found that 92% reported using trial and error to design their most recent reward function (Apdx. A.1). This finding echos the literature: Knox et al. found that, in a survey of RL for autonomous driving, all of the surveyed publications reported designing reward functions by trial and error [103]. Despite the prevalence of trial-and-error reward design, the consequences of this process remain almost completely unexamined by the RL community. It is urgent and crucial for our community to understand the effects of this widespread practice and ultimately to craft specific guidance for *practical* reward design.

When employing trial-and-error, experts often optimize the reward function by manually searching for a reward function that meets the goals of both maximizing the task performance metric and enabling an RL algorithm to learn quickly. This practice raises the question of whether a reward function that is effective with one algorithm can be ineffective with others: can a reward function be overfit to an algorithm? This concern of overfitting raises troubling questions about evaluation in

The content of this chapter largely reproduces the text from Booth, Knox, Shah, Niekum, Stone, Allievi's AAAI 2023 paper: *The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications* [22].

[1] Sutton and Barto disregard their own advice when designing a Dyna-Q+ agent. To encourage exploration, they replace the reward function r with $r + \kappa\sqrt{\tau}$, where κ is a hyperparameter and τ is the number of timesteps [190, p. 168].

RL. Overfitting a reward function to one RL algorithm undermines fair comparisons to another algorithm using the same reward function, especially since the reward function is often also used to measure success.

Consider an ablation study which assesses whether an algorithmic component improves learning. If the reward function is overfit to the algorithm when the component is unablated, observing that learning performance decreases in the absence of the component may merely reflect that the reward function is overfit, giving no clear signal about whether the component is an improvement. Concerns about fair evaluation are already pervasive in RL, and are thought to limit RL's applicability outside of the laboratory [88]. Most such concerns focus on hyperparameters, network architectures, and observed high variance, coupled with the high costs of experimentation [78]. This paper adds an additional perspective: that the oft-overlooked design process behind the reward function must also be considered for fair comparisons.

To assess reward function overfitting, we first conduct computational experiments to test whether certain reward functions enable different RL algorithms and hyperparameters to perform better with respect to the true task performance metric. From these experiments, we find evidence that reward functions can indeed be overfit to a particular discount factor, learning rate, or algorithm: in such cases, changing the discount factor, learning rate, or algorithm significantly diminishes the task metric performance. Across numerous experiments, we find that when we rank reward functions by the learned policies' resultant task metric scores, these rankings are largely *uncorrelated* across experiment variations. Though the idea of reward function overfitting may be unsurprising to seasoned RL experts, the extent of this overfitting problem is nonetheless remarkable.

To learn about the implications of trial-and-error reward design, we also conducted a user study. One goal of this user study was to confirm that our computational experiments' findings correspond to practical effects in realistic RL settings. Specifically, we challenged 30 expert RL practitioners to choose an RL algorithm, hyperparameters, and a reward function to train the best agent they could, as measured by the cumulative task performance metric. The majority of experts overfit

their reward functions to their choice of algorithms or hyperparameters (68%). More alarmingly, many experts also constructed reward functions which failed to encode the task (53%)—meaning these reward functions encoded optimal policies which significantly deviate from the experts' intent, despite these tests being conducted in a simple gridworld environment. We then applied thematic analysis to qualitatively analyze experts' reward-design process, and we discovered that some reward misdesigns stem from mismatched perspectives of what the reward function communicates. For the RL algorithm, reward is an additive component that is used to calculate discounted return—the evaluation metric. Experts instead typically view reward as a direct evaluation of the relative goodness of each state-action pair. This disparity contributes to misdesign as a consequence of trial-and-error reward design.

3.2 Related Work

Reward Shaping

One of the known, common consequences of ad hoc reward design is reward shaping. In reward shaping, the reward function is overloaded to both communicate the underlying performance metric and guide an agent's learning toward a desired policy. Reward shaping can be designed in such a way that the optimal policy is unchanged [151]. However, ad hoc reward shaping is known to be typically *unsafe*—meaning that a shaped reward function is likely to change the optimal solution to a given reinforcement learning task [7, 103]. Our work affirms that ad hoc reward design amplifies this type of misdesign: the resulting optimal policies are often unrecognizable from the expert's known intent. Our work also contributes a new perspective on how trial-and-error reward design results in reward function overfitting, in which reward functions are unintentionally over-engineered for use with a specific algorithm.

Designing Rewards for Fast Learning

Singh et al. asked the philosophical question: where do rewards come from [184]? They established a computational framework for quantifying the performance of re-

ward functions, in which they assess whether 'intrinsic' motivation is helpful—i.e., whether reward functions benefit from rewarding subgoals. We build off of this work, especially the computational framework for assessing reward functions.

Similarly, Sowerby et al. observe that certain reward functions result in faster learning, and they put forth principles of reward design in accordance with this observation [187]. To find fast-learning reward functions, they use linear programming to construct reward functions which meet a correctness criteria of encoding the optimal policy. To test the fastness of learning from these reward functions, they assess how many training steps are needed for a Q-learning agent to converge to the optimal policy. The authors note this work is preliminary and has mostly been tested with a single Q-learning algorithm with fixed hyperparameters. Our work contributes a related perspective: fast learning may not be an intrinsic property of a good reward function, but also a consequence of the paired choice of algorithm and hyperparameters that were used to test that reward function.

AutoRL is another approach, which is gaining increasing traction [155, 52, 37, 212, 204, 162]. AutoRL frames RL as a meta-learning problem, in which the reward function should be learned, perhaps using an evolutionary method. Our work has interesting implications for AutoRL. Currently, these methods usually first optimize a reward function and then fix this function to optimize other RL design choices, such as the neural network architecture. Our work suggests this method—of first fixing the reward function and then optimizing other design choices—may be suboptimal relative to employing a joint optimization strategy.

Inferring Reward Functions

Since specifying reward functions is both known to be hard and requires expertise, many research threads explore how to learn reward functions from intuitive signals like demonstrations [152, 216], preferences [39, 106], and feedback [107, 133]. Inverse reward design is an approach that requires experts to specify reward functions but recognizes that these designed reward functions are only observations about the true goal. As such, inverse reward design works try to infer a true reward function based on

these observations [72, 170]. Our work provides empirical support for this approach, as we find evidence that this assumed human behavior—of designing reward functions as observations and not as true problem specifications—is common in practice. He et al. similarly view reward design is an iterative process [77]. Their work contributes a mechanism for surfacing environments where the reward incentivizes the wrong behavior to the human expert to support them in revising their reward function. Our work reinforces the importance of these types of debugging tools.

3.3 Preliminaries

Reward Function Overfitting

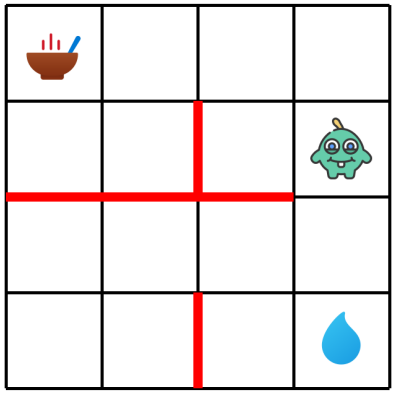
Let $M : \tau \rightarrow \mathbb{R}$ be the true task performance metric. For example, this metric might encode whether the agent reached a goal state or not. Let a **learning context** be a tuple of an RL algorithm, hyperparameter values, and an MDP$\backslash r$; given a reward function, a learning context can be used to train a policy. We claim a reward function r_1 is overfit with respect to one or more learning contexts, $D_1 \sim \mathcal{D}$, if there exists an alternative reward function r_2 such that the task performance metric is optimized over D_1 but not over the larger distribution, $\mathcal{D}$:

$$\begin{aligned} \mathop{\mathbb{E}}_{\tau \sim \pi_{r_1, D_1}}[M(\tau)] &> \mathop{\mathbb{E}}_{\tau \sim \pi_{r_2, D_1}}[M(\tau)] \\ \mathop{\mathbb{E}}_{\tau \sim \pi_{r_1, \mathcal{D}}}[M(\tau)] &< \mathop{\mathbb{E}}_{\tau \sim \pi_{r_2, \mathcal{D}}}[M(\tau)] \end{aligned} \tag{3.1}$$

where D_1 is a set of one or more learning contexts, $D_1 = \{d_1, d_2, \ldots, d_n\}$. This definition is adapted from supervised learning overfitting: the hypothesis space corresponds to the space of possible reward functions and the training and test sets correspond to potential RL algorithms, hyperparameters, and environments [142].

Optimal Reward Functions

Optimal reward functions are related to overfitting: these reward functions are the best performing in a given a learning context. A reward function $r^*_{\mathcal{D}}$ is optimal under



"I am hungry and not thirsty."

Figure 3-2: An example of the Hungry Thirsty domain (4×4 grid). Food and water are each located in a random corner. Red walls are impassable. The current state is abbreviated as $\texttt{H} \wedge \neg\texttt{T}$, which corresponds to the agent being hungry and not thirsty. The 6×6 grid Hungry Thirsty is depicted in [184]. Under the sparse reward function, the optimal policy is for the agent to alternate between traveling to the water and drinking when thirsty, and traveling to the water and eating when not thirsty.

some distribution $\mathcal{D}$ of learning contexts if it maximizes the expected value of learned policies, i.e., $r^*_{\mathcal{D}} = \texttt{argmax}_r \, \mathbb{E}_{\tau \sim \pi_{r,\mathcal{D}}}[M(\tau)]$.

Hungry Thirsty Domain

We use a modified Hungry Thirsty domain [184] as a testbed. This gridworld domain has a fixed time horizon of 200 steps. Food is located in one randomly-selected corner; water in another. Some transitions are blocked by walls (Fig. 3-2). At each timestep, the agent can choose one of six actions: move in a cardinal direction, eat, or drink. The agent's goal is to have sated hunger for as many timesteps as possible. The agent is hungry if and only if it did not eat in the last timestep. However, the agent can only successfully eat if it is co-located with the food and has quenched thirst. On each timestep, the agent stochastically becomes thirsty with 0.1 probability, and only becomes not thirsty if it drinks while co-located with the water. The agent's state is described by its location, as well as two boolean predicates: `H` and `T`, corresponding to hunger and thirst. Time remaining is omitted.

For this task, the performance metric is simply the number of timesteps the agent

has sated hunger: $M(\tau) = \sum_{t=1}^{200} \mathbb{1}(\neg\texttt{H} \in s_t)$. This specific metric can be formulated as a sparse Markovian reward function, $r(s, a, s') = \mathbb{1}(\neg\texttt{H} \in s)$.[2] Under the optimal policy for this reward function, the agent alternates between navigating to the water or drinking when thirsty, and navigating to the food or eating when not thirsty. While it is often possible for RL algorithms to learn with this sparse reward function, shaped reward functions that reward the not thirsty ($\neg\texttt{T}$) subgoal or punish time spent hungry ($\texttt{H}$) let many RL algorithms solve this domain faster and more easily. These properties make this domain an interesting testbed for studying reward design.

For our experiments, all reward functions take the form:

$$r(\texttt{H} \wedge \texttt{T}) = a \qquad r(\texttt{H} \wedge \neg\texttt{T}) = b$$
$$r(\neg\texttt{H} \wedge \texttt{T}) = c \qquad r(\neg\texttt{H} \wedge \neg\texttt{T}) = d$$

where $a, b, c, d \in \mathbb{R}$. Since there are no reward components for location, opportunities for shaping—and, thereby, overfitting—are limited but still possible. For shorthand, we write reward functions as $[a, b, c, d]$. We say reward functions encode the task when the optimal policy matches the optimal policy derived from the sparse reward function. Singh et al. found the highest-performing reward function to be $[-0.05, -0.01, 1.0, 0.5]$ for a continuing version of this domain [184]; this reward function is dense and notably rewards drinking water as a subgoal.

When conducting large experiments, we assign each of a, b, c, and d to a value from the set: $\{\pm 1, \pm 0.5, \pm 0.1, \pm 0.05, 0\}$, the same values used in prior art [184]. Reward functions that meet the following criteria trivially do not encode the task and are thus excluded: $r(\texttt{H} \wedge \neg\texttt{T}) \geq r(\neg\texttt{H} \wedge \texttt{T})$ and $r(\texttt{H} \wedge \neg\texttt{T}) \geq r(\neg\texttt{H} \wedge \neg\texttt{T})$. Such reward functions encode an incorrect optimal policy of navigating to the water and consistently drinking. This filtering leaves 5,196 reward functions.

[2]Such reformulation as a Markovian reward function is not universally possible across all task performance metrics.

3.4 Computational Experiments

We first assess overfitting in reward functions by conducting large-scale performance comparisons, in which we measure a learning context's ability to optimize the task performance metric given a reward function. As an intuitive example, we speculate that when using an RL algorithm with a high learning rate, a high magnitude reward function might be less likely to lead to convergence within a fixed training duration than a lower magnitude reward function.

To study this relationship between reward function design and hyperparameters empirically, we assess the mean task performance metric accumulated over all 200-timestep episodes of training achieved by learning with different reward functions across varied Q-learning hyperparameters: γ (the environment discount factor) and α (the learning rate). While γ is formally defined as a parameter of the environment, and not the learning algorithm, it is typically selected to construct a viable horizon for applying an RL algorithm [92] and can thus be equally considered a hyperparameter of the learning algorithm. We additionally study whether the reward functions are overfit to the learning algorithm itself by comparing performance metrics with several deep RL methods: A2C [143], DDQN [144], and PPO [180] in the 6×6 Hungry Thirsty domain. Unless otherwise specified, we train 10 agents per experimental setting.

H1: Reward functions that are effective in one learning context can be ineffective in another. There exist two different learning context samples (D_1 and D_2) and a reward function r_1 such that r_1 achieves high cumulative performance (as measured by the true performance metric) when tested with D_1 but low cumulative performance with D_2. In formal terms, there exists a reward function r_1 such that

$$\mathop{\mathbb{E}}_{\tau \sim \pi_{r_1, D_1}} [M(\tau)] > \beta_1 \text{ and } \mathop{\mathbb{E}}_{\tau \sim \pi_{r_1, D_2}} [M(\tau)] < \beta_2,$$

where β_1 is some high threshold (e.g., performing among the top 25% of reward functions when tested with D_1) and β_2 is some low threshold (e.g., performing among the bottom 25% of reward functions when tested with D_1).

H1 tests whether some reward functions enable successful learning in some learning contexts but not in others. In other words, this hypothesis assesses whether reward function overfitting can occur. Although the learning contexts in D_2 could be chosen adversarially—i.e., to include an RL algorithm incapable of learning—we assume all learning contexts in D_2 aim to maximize expected return and are generally capable of doing so.

H2: Reward functions that are optimal in one learning context can be suboptimal in another. Given a reward function $r^*_{D_1}$ that is optimal with respect to D_1, a different reward function $r^*_{D_2}$ may be optimal with respect to D_2. In formal terms, there exist two learning context samples D_1 and D_2 such that $r^*_{D_1} \neq r^*_{D_2}$.

H2 tests tests whether the reward functions which are found to be best-performing are consistently best-performing across multiple learning contexts. As in **H1**, we assume that all considered learning contexts aim to maximize expected return and are generally capable of doing so.

H3: The performances of different reward functions are uncorrelated across learning contexts. If reward functions are ranked by their average cumulative performance metric scores (e.g., $r_a > r_b > r_c > ...$ for a learning context sample D_1), the ranked reward functions from D_1 will be uncorrelated with the ranked reward functions from a different sample, D_2. In formal terms, for some set of reward functions $r_1, r_2, \ldots, r_n$, $\mathbb{E}_{\tau \sim \pi_{r_i, D_1}}[M(\tau)]$ will be *uncorrelated* with $\mathbb{E}_{\tau \sim \pi_{r_i, D_2}}[M(\tau)]$ for $1 \leq i \leq n$.

H3 examines the commonality of reward function overfitting. If this phenomenon is rare, correlation across learning contexts should be high. If it is common, correlation should be low. Of these hypotheses, confirmation of **H3** is most concerning as it indicates extensive reward function overfitting.

3.4.1 Overfitting to Hyperparameters

We first assess whether reward functions can be overfit to either the discount factor, γ, or the learning rate, α. For this experiment, we use a Q-learning agent trained over

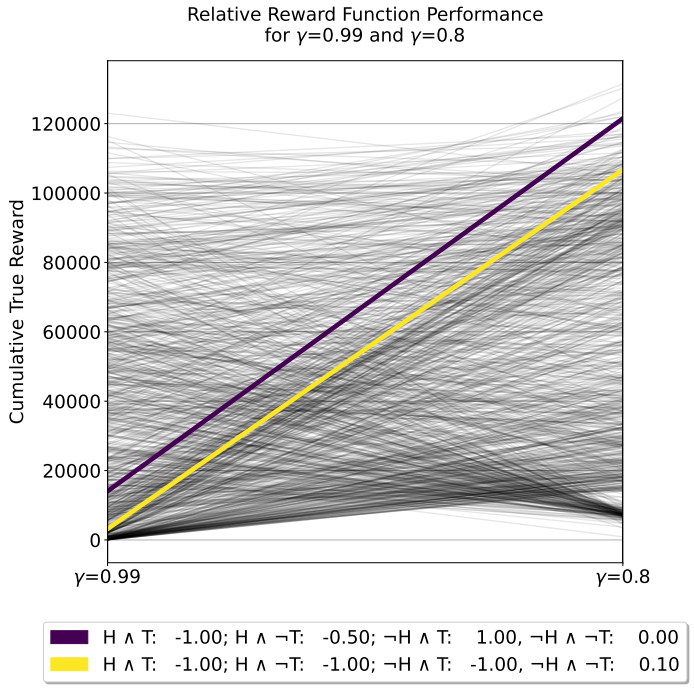


Figure 3-3: A parallel coordinate plot showing the paired rankings of reward functions. Each line corresponds to a reward function, with cumulative performance averaged over 10 independently-trained agents. The many intersections portray the uncorrelated nature of these rankings. The two reward functions with the largest cumulative difference in performance are highlighted. These reward functions result in low cumulative performance when $\gamma = 0.99$, but high performance when $\gamma = 0.8$. See Apdx. A.3 for more examples.

2000 episodes. For evaluating overfitting to the discount factor, we vary γ for each learning context: $\gamma = 0.99$, $\gamma = 0.8$, and $\gamma = 0.5$. For evaluating overfitting to the learning rate, we consider $\alpha = 0.05$ and $\alpha = 0.25$. The standard hyperparameters are described in Apdx. A.4. We average performance, as measured by the cumulative true reward, over 10 trials to account for stochasticity stemming from the environment or from the learning process (i.e., randomized weights).

H1: Reward functions that are effective in one learning context can be ineffective in another. For all experiments, we find some reward functions which result in policies which achieve high task performance scores when trained with one learning context but low task performance scores when trained with a different learning context. Some such reward functions are highlighted in Fig. 3-3 and Apdx. Fig. A-1. To assess whether these differences are not just a consequence of stochastic policy

Reward Function	Experiment	Hoeffding Bound	p-value
[-1.0, -1.0, -1.0, 0.1]	$\gamma = 0.99$	[4,965; 20,234]	< 0.01
	$\gamma = 0.8$	[86,653; 101,922]	
[-0.1, 0.2, 0.5, 1.0]	DDQN	[94,790; 182,944]	< 0.01
	A2C	[-29,040; 59,114]	

Table 3.1: A comparison of reward function performance assessed over 1000 trials (Q-learning) or 30 trials (deep RL methods). Performance is assessed with the Hoeffding Bound, which is akin to a confidence interval, and a Mann Whitney U-test. This data confirms that the same reward function can lead to very different performance with different hyperparameters or algorithms.

learning, we re-ran these experiments with the reward functions which resulted in maximally different true performance for 1000 additional trials. We then computed the 90% Hoeffding Bound [80], which bounds the average cumulative task performance metric across trials with 90% probability, and we separately performed a Mann Whitney U-test [148] to assess whether the mean cumulative true task performance values were drawn from the same underlying distribution. We find, in all cases, we can reject the null hypothesis that these underlying distributions are the same as the observed differences are all statistically significant ($p < 0.05$). We conclude that, across varied hyperparameters, reward functions that are effective in one learning context can be ineffective in another. See Tab. 3.1 and Apdx. Tab. A.1.

H2: Reward functions that are optimal in one learning context can be suboptimal in another. Across each pair of tested learning contexts, we find that the best-performing reward function differs (Apdx, Fig. A-2). We further confirm this finding by fixing an experimental learning context and assessing whether the best-performing reward function for that learning context outperforms the top-3 reward functions from a different experimental learning context, testing the cumulative task performance metric for each reward function over 1000 trials. For example, the best-performing reward function function for $\gamma = 0.99$ was $[-0.05, -0.05, 0.5, 0.5]$, which outperformed the best-performing reward function for $\gamma = 0.5$, $[-1.0, -1.0, 0.0, 1.0]$. We then compute the 90% Hoeffding Bound for the mean cumulative task performance metric, and we separately conduct a Mann Whitney U-test to assess whether

# of Reward Fns	D_1	D_2	τ_b	p-value
5196	$\gamma = 0.99$	$\gamma = 0.8$	0.07	< 0.01
	$\gamma = 0.99$	$\gamma = 0.5$	0.04	0.07
	$\gamma = 0.8$	$\gamma = 0.5$	0.12	< 0.01
	$\alpha = 0.25$	$\alpha = 0.05$	0.11	< 0.01
107	PPO	A2C	0.25	0.01
	PPO	DDQN	-0.04	0.62
	PPO	QLearn	0.13	0.08
	A2C	QLearn	-0.08	0.29
	A2C	DDQN	-0.01	0.87
	DDQN	QLearn	-0.06	0.41

Table 3.2: Kendall's τ_b correlation over the 5196 tested reward functions for hyperparameter experiments and 107 tested reward functions for algorithm experiments. $\tau_b \in [-1, 1]$. $|\tau_b| < 0.1$ indicates the variables are uncorrelated; $|\tau_b| < 0.2$ indicates a weak correlation. In our experiments, H3 is supported with low τ_b values (even with high p-values). Almost all comparisons are either uncorrelated or weakly correlated; H3 is supported for all experiments except PPO vs. A2C. This data confirms that the choice of reward function is highly sensitive for RL algorithm performance.

the distribution of the best-performing reward function's performance is greater than that of the alternative tested reward function (which is best-performing for a different experimental condition). We find that we can reject the null hypothesis that these reward functions result in equal performance distributions in 16 of 18 experiments ($p < 0.05$). In general, the best-performing reward function for one learning context outperforms the top-3 reward functions for another learning context. See Apdx. Tab. A.2.

H3: The performances of different reward functions are uncorrelated across learning contexts. We compute Kendall's tau rank correlation to assess **H3**. This measures the strength and direction of the monotonic association between rankings, without considering the difference in magnitude of the performance metric since some learning contexts may be consistently 'better' or 'worse' in terms of raw performance. We used a nonparametric test, since the cumulative scores were not found to be normally distributed over many trials. In this setting, the null hypothesis is that two random variables are independent (i.e., $\tau_b = 0$). **H3** is supported with low τ_b values, even with high p values. Generally, the closer τ_b is to 0, the more samples are

needed to show significance. We find that reward function performance is **uncorrelated** ($|\tau_b| < 0.1$) or **weakly correlated** ($|\tau_b| < 0.2$, see Table 3.2) in all discount factor and learning rate experiments. We conclude that performance across varying hyperparameters is sensitive to reward function choice.

From these experiments, we find consistent evidence that reward functions can be overfit to RL hyperparameters.

3.4.2 Overfitting to RL Algorithms

For the Section 3.4.1 Q-learning experiments with varied hyperparameters, we trained 5196 agents, 10 times each. The protocol for generating these reward functions is described in Section 3.3. In the deep RL setting, this scale of training is infeasible because training each agent takes between 3 and 11 minutes (Apdx. A.6). To test overfitting in this setting, we instead consider a restricted set of reward functions to reduce the computational burden. We source these reward functions from the user study; specifically, we consider the set of unique reward functions that experts handcrafted at any point during their sessions and that also encode the desired optimal policy in easy environment configurations (Section 3.5). In total, we analyze 107 reward functions in this deep RL setting. For each reward function, we train A2C, DDQN, PPO, and Q-learning agents. We train each agent over 5000 episodes, and average performance over 10 trials.

H1: Reward functions that are effective in one learning context can be ineffective in another. We again find that every experiment variation uncovers reward functions which enable successful learning in one experimental learning context, but not the other. For example, the reward function $[-0.1, 0.2, 0.5, 1]$ achieved a high mean cumulative performance metric score of 232055 for DDQN, but a low mean cumulative score of 107—indicative of never learning the optimal policy—for A2C. We then ran this specific test an additional 30 times, and found evidence that we can again reject the null hypothesis that these true task performances were drawn from the same underlying distribution ($p < 0.05$). See Table 3.1 and Appendix Fig. A-3,

in which the reward functions that achieve maximally different performance metric measures are highlighted.

H2: Reward functions that are optimal in one learning context can be suboptimal in another. In 5 of 6 experiments varying the RL algorithm, the optimal reward functions differ. This was only not true is in the PPO and A2C comparisons. In this case, the true performance metric function itself ($[0, 0, 1, 1]$) is the optimal reward function for both algorithms. See Appendix Fig. A-3.

H3: The performances of different reward functions are uncorrelated across learning contexts. Again using Kendall's τ_b for assessment, we mostly find evidence that reward functions' cumulative true performance metric scores are **uncorrelated** when varying the RL algorithm ($|\tau_b| < 0.1$, see Table 3.2). Specifically, we find the PPO vs. DDQN, A2C vs. Q-learning, A2C vs. DDQN, and DDQN vs. Q-learning agents to be mutually uncorrelated. We find evidence of **weak correlation** between PPO and Q-learning ($|\tau_b| < 0.2$). Lastly, we find evidence of **some correlation** ($\tau_b = 0.25$) for the PPO vs. A2C comparison. Statistical significance—which allows us to reject the null hypothesis that the two random variables are independent—is generally not established due to the reduced sample size.

From these experiments, we find evidence that reward functions can be overfit to RL algorithms.

3.5 Expert Human Subject Experiments

To assess how experts design reward functions and whether this problem of reward function overfitting caries over to realistic settings, we conduct a controlled observation study.

Study Population

We conducted 2 pilot studies, followed by 30 studies with expert participants drawn from four US research universities (R1). To qualify as an expert, participants were

required to meet one or more of the following criteria: (1) have experience conducting research on RL methods; (2) have used RL methods in research; or (3) have passed a class which covered reinforcement learning in depth. Of the 30 participants, 1 was a post-doctoral scholar, 17 were PhD students, 5 were research-based master's students, and 7 were advanced undergraduates. Participants are each assigned a study ID, ranging from **P0** to **P29**.

Study Protocol

The study session took one hour and was primarily conducted in-person (19 of 30 sessions). Participants were compensated $40 USD in the form of an Amazon giftcard. The study used a Jupyter notebook, in which participants were required to select a reward function, algorithm, and hyperparameters to train an agent to solve the Hungry Thirsty task (Apdx. A.2). Participants were asked to speak aloud as they worked, and the experimenter took detailed notes. The experimenter occasionally asked open-ended questions (such as "what are you trying to do now?") to prompt the participant to continue speaking. Five minutes before the end of the session, participants were asked to submit their best configuration consisting of some reward function r_i and some algorithm and hyperparameter selection, D_i. Afterwards, participants were asked to answer five structured questions (Apdx. A.2.3).

The participants were informed that the research team would independently train an agent using their submitted reward function, algorithm, and hyperparameters, and that if this trained agent performed in the top ten across all participants' agents in terms of cumulative performance, they would receive a $10 USD bonus, again in the form of an Amazon giftcard. Participants were required to train at least three different agents—though the experimenter explicitly noted that they could simply re-train the same agent three times to meet this requirement.

The first 12 participants used a 6×6 grid for the Hungry Thirsty domain. After observing participants struggling to solve this task, we reduced the size of the grid to 4×4 for the remaining 18 participants. The study was approved by the IRB.

Experts Often Design Invalid Reward Functions

The Hungry Thirsty domain has harder and easier environment configurations. In total, there are 12 different configurations, which correspond to different placements of the food and water. In a 6×6 grid, the food and water can either be located 5 steps apart in the best case or 16 steps apart in the worst case. In a 4×4 grid, these distances are 3 and 9 steps in the best and worst cases, respectively. As in the original version of Hungry Thirsty [184], the locations of the food and water are randomly resampled each time the user trains a new agent, but remain consistent throughout the lifetime of the agent. In this study, the user is tasked with designing a reward function which is invariant to any choice of environment configuration.

To determine whether a reward function is valid for a given task configuration (i.e., for fixed food and water positions), we empirically assess whether a policy—learned with value iteration—is the same as the optimal policy under the sparse reward function. Specifically, we use value iteration (with $\theta = 0.01$ as the end criteria) to solve for an approximately optimal policy using the sparse reward function and $\gamma = 0.99$. We then use value iteration to solve for a policy using the user's submitted reward function and choice of γ. We run 100 test episodes for each agent with a fixed random seed, and use the average cumulative undiscounted task performance metric for comparison. If the policy learned with the user's reward function has the same cumulative undiscounted task performance as the policy learned with the sparse reward function, we consider it valid. If the user's reward function is valid for all environment configurations, we say it encodes the task.

The majority of participants (83%) successfully selected reward functions which were valid with the easier placements of the food and water on adjacent corners (10 of 12 in the 6×6 setting; 15 of 18 in the 4×4 setting). However, only 47% of participants selected reward functions which were valid when the food and water are maximally distant, at opposite corners (4 of 12 in the 6×6 setting; 10 of 18 in the 4×4 setting). For example, **P23**'s reward function $[-0.05, 0.5, 0.5, 1.0]$ is valid in the easier adjacent case but not the opposite-corners case, because when the food and

water are maximally distant the optimal policy causes the agent to remain in the state $\texttt{H} \wedge \neg\texttt{T}$. Finding this form of misdesign, where reward functions are only valid in some environment configurations, adds support to the research pursuit of inverse reward design methods [72], which is built upon the perspective that reward functions should be considered an observation about the expert's intended reward function and not as a perfect specification.

Experts Overfit Reward Functions to Algorithms

Even when experts wrote reward functions which encode the task, they typically continued to edit their reward functions. Each expert tried a sequence of reward functions $r_1, r_2, \ldots, r_n$ and finally settled on some reward function r_i where $i \in [1, n]$. The user evaluated each of these reward functions alongside potentially-changing algorithms and hyperparameters, $D_1, D_2, \ldots, D_n$ and settled on some choice D_i. Because every aspect of the user's solution may be changing simultaneously, this setting is messier and harder to evaluate than the purely-computational setting. To evaluate overfitting, we test all of the user's reward functions with standard implementations for DDQN, PPO, and A2C and fixed hyperparameters (Apdx. A.4). We discard the user's algorithms (i.e., D_i) and exclusively test the user's reward functions.

In this setting, we define overfitting to have occurred if one or more of the user's tested reward functions (r_j, where $j \in [1, n]$ and $r_j \neq r_i$) significantly outperforms their final selection with respect to one or more of the three tested RL algorithms. We define this performance difference threshold to be 20000, accumulated over 5000 training episodes and averaged over 10 trials. This overfitting assessment is different from the computational setting, which requires comparing the rankings and not absolute performance between different reward functions. Since each user tried only a small handful of reward functions (on average, 4.1 unique reward functions), these rankings are less meaningful.

Of the users who tried multiple reward functions and submitted a best-case-valid reward function, 68% (15 of 22) overfit their reward functions. For example, participant **P20** tried the reward function $[-0.1, 0.1, -0.1, 1]$, which achieved a mean

cumulative performance of 138,092 using DDQN. In their final selection, this user instead chose the reward function $[-5, 15, 5, 100]$, which achieved a mean cumulative performance of 1,031 using DDQN. We include an alternative metric for overfitting in the user study in Apdx. A.5.

Assessing the Design Process with Thematic Analysis

To analyze not just experts' reward design outcomes, but also their design process, we applied qualitative analysis in the form of *thematic analysis* [27, 82]. Thematic analysis is a system for extracting patterns from qualitative data by systematically coding and analyzing transcripts. To perform thematic analysis, every statement of each transcript is first assigned a summary (also known as a code). Each of these summaries is then further distilled into a detailed, low-level theme. These low-level themes are finally distilled into high-level themes. Thematic analysis is generally considered successful if the resulting themes are consistent and coherent, and describe the data they incorporate well. In such cases, the extracted themes provide insight into the unstructured data. In our application of thematic analysis, the first summary step of this process generated 990 codes. The second step generated 212 low-level themes. And, finally, the third step resulted in the extraction of 10 high-level themes. We include the full analysis in the supplementary material. Here, we discuss these themes and their implications for reward design.

Experts' Approaches to RL and Reward Design

Thematic analysis showed that experts use one or more of the following strategies when tasked with crafting and solving an RL task: folklore-based, intuition-based, trial-and-error-based, hypothesis-based, random-based, or reason-based. For example, **P25** declared, "I've heard that reward scaling is pretty important", and this quote is an example of using a folklore-based process. Concerning this same parameter choice, **P27** declared, "The reward scaling factor must be very large, I think, since you might only see little food," and this quote is an example of using a reason-based process. Experts often switched between two or more strategies.

Trial-and-Error Reward Design is Typical

This user study was designed to be naturalistic; participants could choose to focus primarily on any combination of the three axes of choice, between specifying the reward function, algorithm, and hyperparameters. Conventional reward design wisdom suggests that experts should try to align a reward function as closely as possible with the task completion criteria, and should only adjust the reward function if it is found to not encode correct measurements of task outcomes.

93% of experts tried at least two reward functions (only **P2** and **P4** stuck to a single reward function). Experts tried 4.1 unique reward functions on average. In this study setting, shaping was unnecessary: any of the available algorithms could learn from the sparse reward function. Despite this, and almost all users (97%) shaped their reward functions. This finding is compelling: even in the absence of a need to shape, experts gravitate towards doing so.

Analyzing study transcripts, we find that half of experts (**P5-13, P15-17, P20, P23, P27-28**) explicitly noted a perceived error at least once before modifying their reward function (thus employing a reason-based process). For example, **P5** stated: "I realized I'm penalizing the $\texttt{H} \wedge \texttt{T}$ state too much, because the agent knows it will be penalized on the way back [to the water]." In contrast, some experts indicated they were relying on trial and error: **P28** stated, "The worst possible state to be in is $\texttt{H} \wedge \texttt{T}$, so I'm going to assign -1. The best possible state to be in is $\neg\texttt{H} \wedge \neg\texttt{T}$, so I'm assigning that to 1. $\texttt{H} \wedge \neg\texttt{T}$ is not particularly as bad as $\texttt{H} \wedge \texttt{T}$... Setting that to -0.25. Reward for $\neg\texttt{H} \wedge \texttt{T}$: not too close to -1; I'll just assign some arbitrary small value."

A Common Misdesign Cause: Weighing State Goodness

Weighing state goodness to design a reward function was a recurring low-level theme. Most experts (83%) stated something to the effect of: "It's best to be $\neg\texttt{H} \wedge \neg\texttt{T}$, so I'll set that to the max, 1. Being $\neg\texttt{T}$ is better than being $\neg\texttt{H}$. Worst is at $\texttt{H} \wedge \texttt{T}$; setting that to -1" (**P25**; this statement corresponded to their invalid reward function [-1.0, 0.3, -0.35, 1.0], for which the optimal policy is to remain drinking water indefinitely).

This reward design practice—of using the reward function to rank the goodness of immediate states and/or actions, applying a myopic design strategy without assessing how the reward function will be used as an optimization target for computing expected discounted return—often led to reward misdesign, as it did for **P25**.

Though less often, some experts did recognize the importance of reward accumulation and state visitation frequency (another recurring low-level theme). For example, **P23** stated "A positive reward for $\texttt{H} \wedge \neg\texttt{T}$ is not the way to go. A combination with a negative reward for $\texttt{H} \wedge \texttt{T}$ makes it worse, since it would rather accumulate positive rewards at the water instead of searching for food." This design process—of considering summed reward, which aligns with the RL optimization objective—was relatively rare (i.e., 30% of experts noted something to this effect). From this qualitative analysis, we found this lack of emphasis on reward accumulation and expected discounted return to be the main cause of explicit reward misdesign, wherein reward functions were invalid and did not correctly encode the task.

This observation—that humans assume a myopic interpretation of reward functions, in which reward accumulation is largely ignored—has previously been observed in another setting of reinforcement learning from human feedback. [108] discovered that when learning a reward function from non-expert human feedback, humans adopt a similarly myopic teaching strategy. Finding that this myopic interpretation is echoed across both non-expert and expert users can inform future efforts to support humans in designing reward functions, and can help reinterpret how humans' reward functions should be used for optimization.

3.6 Limitations

While we studied the Hungry Thirsty domain in depth in this work, we only evaluated reward design practice in this one domain. Hungry Thirsty is a rich testbed for assessing reward design practice, but understanding this practice across multiple domains with diverse properties equally deserves attention. This domain in particular allows us to assume the existence and specification of a true task performance

metric, but in many circumstances, specifying such a metric is itself a challenging problem. In such cases, the methodology we use to study reward design would not readily reapply.

Another limitation of this work concerns the definition of reward function overfitting. Our definition omits a temporal aspect to the distribution ($\mathcal{D}$, consisting of algorithms, hyperparameters, and tasks) that makes samples from $\mathcal{D}$ dependent (i.e., not i.i.d.). For example, if an expert has tested a reward function r with one RL algorithm and a set of hyperparameter values, we suspect such an expert is more likely to next test the reward function r with the *same* algorithm and different hyperparameter values than with a different RL algorithm. This temporal component is omitted from our overfitting definition—as it similarly tends to be in the supervised learning setting—and future work could explore the consequences of this omission.

3.7 Discussion

Despite the prevalence of trial-and-error reward design, the implications of this practice remain underexplored. In this first analysis of the consequences of this practice, we identify two problems: reward function overfitting and the frequent design of invalid task specifications. In overfitting, reward functions are designed with respect to a fixed algorithm or hyperparameter set, and the resulting reward functions bias toward better learning given these design choices. This finding contributes to concerns around reproducibility in RL: we find the performance of the reward function is often dependent on the choice of algorithm. For RL practitioners, one takeaway from this work is that the reward function—like the discount factor [92]—should be defined twice: once to specify the true problem as part of the MDP, and once as a form of hyperparameter for the RL algorithm to facilitate learning. This separation accommodates the need to design a reward function for successful learning while also supporting fair evaluations.

In addition to overfitting, we find that ad hoc trial-and-error reward design leads to misdesign as invalid task specifications, wherein experts design reward functions which

fail to encode the desired task even in a simple gridworld domain. One candidate cause for this misdesign is that experts typically adopt a myopic interpretation of reward, and this interpretation is at odds with the RL objective of optimizing cumulative and perhaps discounted rewards. Given this finding, one direction for future work would assess the systemic errors humans make when designing reward functions, and try to construct better mechanisms for inferring the humans' true intent given these systemic errors. Such a mechanism could build off of inverse reward design [72].

While there is great optimism around the flexibility of reward as the optimization target for learning [182], this paper contributes to mounting evidence that most people are ineffective reward designers in *current* practice [7, 111, 103]. As future work, we assert that the community should also explore mechanisms to support humans—including experts!—in this reward design endeavor. Specifically, one could develop guidance for the human reward designer's process such that it more directly reflects the RL optimization target of expected discounted return. Additionally, it is worth exploring whether incorporating explanation mechanisms can improve reward design outcomes (for example, by assisting experts in assessing the contributions of decomposed reward components [94]).

Alternative models of reward should also be considered and evaluated both for their propensity for overfitting and for their propensity for other forms of misdesign. Reward machines [89] and hybrid reward architectures [200] are two such compelling candidates. In reward machines, reward functions are described as a type of finite state machine instead of directly as a function. While reward machines may elicit feature engineering and are thus taboo in RL, humans may be better able to design reward functions which correctly encode a task if they use sufficient structure for guiding the design process. In hybrid reward architectures, the reward function is decomposed into n different reward functions, each of which is then used to optimize a policy. These policies are subsequently aggregated into a single policy. These methods both induce supporting structures for designing reward functions, and this support may help humans write better reward functions with lowered propensity for overfitting or other forms of misdesign.

Chapter 4

Inspect: Classifiers

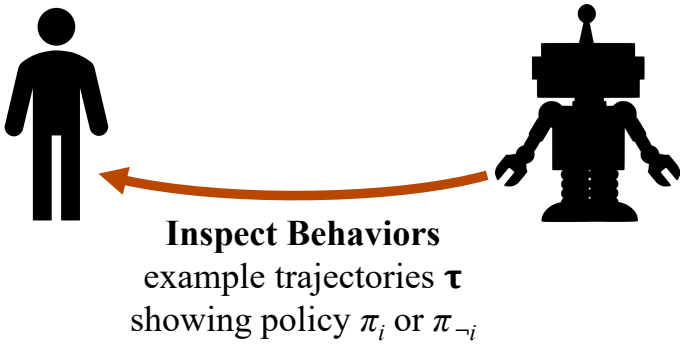


Figure 4-1: This chapter focuses on inspecting behaviors, particularly for neural networks and classifiers. This technique is adapted for robot controllers in Chapter 5.

Inspecting the behaviors of AI systems is critical for supporting humans in finding the flaws in their specifications. Inspecting behaviors is the second interaction component of the interactive system described in Figure 1-1 and highlighted above in Figure 4-1. To begin designing methods in service of inspecting AI system behaviors, this dissertation temporarily suspends the focus on sequential decision-making systems, and focuses first on inspecting the learned behaviors of classification models such as neural networks. This setting is simpler, since classifiers are not subject to the temporal and spatial constraints of sequential decision-making settings. This chapter introduces a method for inspecting the learned behaviors of classifiers; Chapter 5 modifies this method to inspect the learned behaviors of RL and other robot controllers.

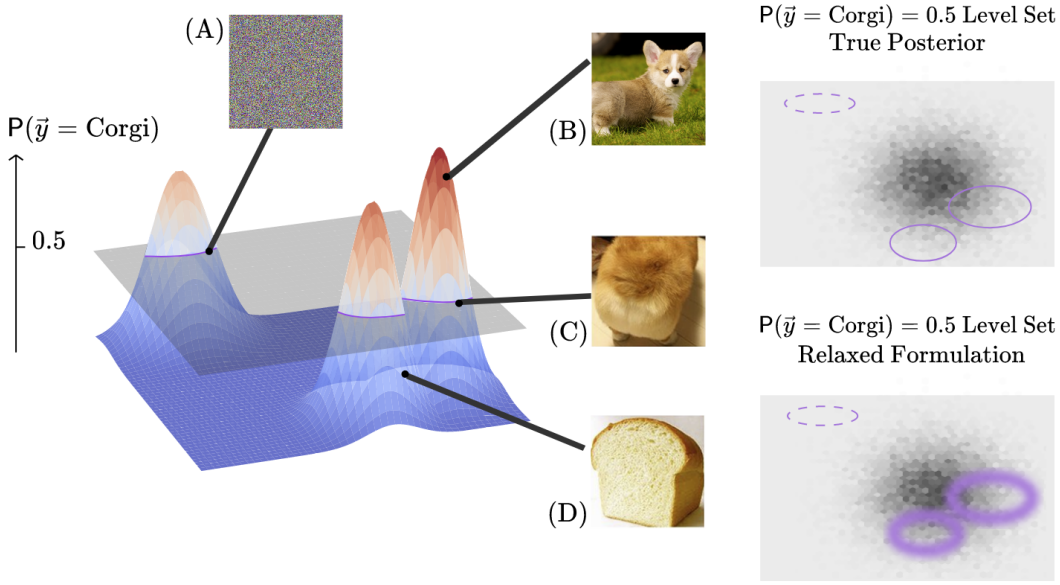


Figure 4-2: Given a neural network which classifies images as either "Corgi" or "Bread," we generate *prediction level sets*, or sets of examples which trigger a target prediction confidence (e.g., $\vec{p}_{\text{Corgi}} = \vec{p}_{\text{Bread}} = 0.5$). Perturbing an arbitrary image to trigger the target confidence is one way of finding such examples, as shown in (A). However, such examples give little insight into the typical model behavior because they are unrealistic and unlikely. For more insight, Bayes-TrEx explicitly considers a data distribution (gray shading on the bottom plots) and finds in-distribution examples in a particular level set (e.g., likely Corgi (B), likely Bread (D), or ambiguous between Corgi and Bread (C)). Bottom left: the classifier level set of $\vec{p}_{\text{Corgi}} = \vec{p}_{\text{Bread}} = 0.5$ overlaid on the data distribution. Example (A) is unlikely to be sampled by Bayes-TrEx due to near-zero density under the distribution, while example (C) is likely to be sampled. Bottom right: Sampling directly from the true posterior is infeasible, so we relax the formulation by "widening" the level set. By using different data distributions and confidences, Bayes-TrEx can uncover examples that invoke various model behaviors to improve model transparency.

4.1 Introduction

Debugging, interpreting, and understanding neural networks can be challenging [45, 128, 156]. Existing interpretability methods include visualizing filters [210], saliency

This chapter's content largely reproduces the text from Booth, Zhou, Shah, and Shah's AAAI 2021 paper: *Bayes-TrEx: a Bayesian sampling approach to model transparency by example* [22].

maps [183], input perturbations [172, 131], prototype anchoring [124, 36], tracing with influence functions [110], and concept quantification [63]. While some methods analyze intermediary network components such as convolutional layers [17, 157], most methods instead explain decisions based on specific inputs. These inputs are typically selected from the test set, which may lack examples that lead to highly confident misclassifications or ambiguous predictions. Thus, it may be challenging to extract meaningful insights and attain a holistic understanding of model behaviors by using only test set inputs. Finding and analyzing inputs that invoke the gamut of model behaviors would improve *model transparency by example*.

To create new examples beyond the scope of the test set, BAYES-TREX takes a data distribution—either manually defined or learned with generative models—and finds in-distribution examples that trigger various model behaviors. BAYES-TREX finds examples with target prediction confidences $\vec{p}$ by applying Markov-Chain Monte-Carlo (MCMC) methods on the posterior of a hierarchical Bayesian model. For example, Fig. 4-2 shows a Corgi/Bread classifier. For different $\vec{p}$-level set targets (e.g., $\vec{p}_{\text{Corgi}} = \vec{p}_{\text{Bread}} = 0.5$), BAYES-TREX can find examples where the model is highly confident in the Corgi class, in the Bread class, or ambiguous between the two. We use BAYES-TREX to analyze classifiers trained on CLEVR [93] with a manually defined data distribution, as well as MNIST [119] and Fashion-MNIST [206] with data distributions learned by variational autoencoders (VAEs) [101] or generative adversarial networks (GANs) [66].

BAYES-TREX can aid model transparency by example across several contexts. Each context requires a different data distribution and a specified prediction confidence target. For example, BAYES-TREX can generate *ambiguous* examples to visualize class boundaries; *high-confidence misclassification* examples to understand failure modes; *novel class* examples to study model extrapolation behaviors; and *high-confidence* examples to reveal model overconfidence (e.g., in domain-adaptation). In all of these use cases, the discovered examples can be further assessed with existing local explanation techniques such as saliency maps (Fig. 4-3).

The main current alternative to BAYES-TREX is to inspect a model by using

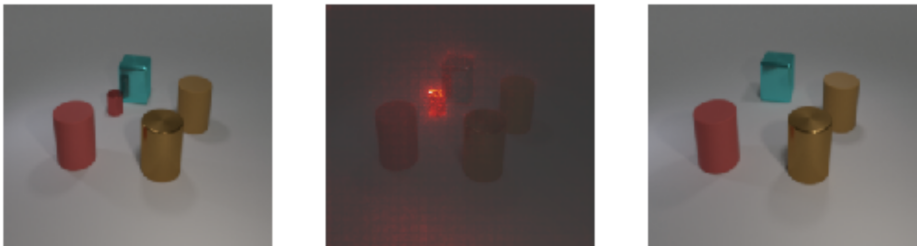


Figure 4-3: BAYES-TREX finds a CLEVR scene which is incorrectly classified as containing a sphere. The generated example (left) is composed of only cylinders and cubes, but the classifier is 97.1% confident this scene contains one sphere. The SmoothGrad [186] saliency map highlights the small red cylinder as the object that is confused for a sphere. When we remove it, the classifier's confidence that the scene contains one sphere drops to 0.1%.

test set examples. As a baseline comparison, we search for highly confident misclassifications and ambiguous examples in the (Fashion-)MNIST and CLEVR test sets. We find few such test set examples meet these constraints, and the majority of these can be attributed to mislabeling in the dataset collection pipeline rather than misclassification by the model. In contrast, BAYES-TREX consistently finds more highly confident misclassified and ambiguous examples, which enables more flexible and comprehensive model inspection and understanding.

4.2 Related Work

4.2.1 Model Transparency

Broadly, transparency is achieved when a user can develop a correct understanding and expectation of model behavior. One common technique for developing transparency is the test set confusion matrix: this matrix expresses the classifier's tendency of mistaking one class for another. Other transparency methods try to "open" black-box models—for example, by visualizing convolutional filters through optimization [50, 157] or image patches [17]. Like BAYES-TREX, other transparency methods communicate model behaviors through examples—for example, with counterfactuals [10, 96] or with student-teacher learning examples [164].

Some transparency methods aim to explain a model's response to an individual

input. For example, saliency maps compute a heat map over the input that represents the importance of each pixel [183, 210]. Importantly, these input-based methods require a two-stage pipeline: finding interesting inputs → explaining the model responses (e.g., with saliency maps). Current efforts are focused on the second stage with inputs simply retrieved from the test set. To the best of our knowledge, Bayes-TrEx is the first work dedicated to the first stage of finding interesting inputs. The examples uncovered by Bayes-TrEx can be used with any input-based method for further analysis (Fig. 4-3 and App. B.11).

4.2.2 Model Testing

TensorFuzz [156] is a fuzzing test framework for neural networks which finds inputs that achieve a wide coverage of user-specified constraints. TensorFuzz is similar to Bayes-TrEx in that both methods aim to find examples that elicit certain model behaviors. While TensorFuzz is designed to find *rare* inputs that trigger edge cases such as numerical errors, Bayes-TrEx finds common, in-distribution examples. As such, Bayes-TrEx is more suitable to help humans develop a correct mental model of the classifier. Scenic [56] is a domain-specific language for model testing by generating failure-inducing examples. While Bayes-TrEx is in part inspired by Scenic, its formulation is more flexible since it is not a constrained domain-specific language.

4.2.3 Natural Adversarial Examples

One Bayes-TrEx use case is uncovering high-confidence classification failures in the data distribution. This idea is related to, but different from, natural adversarial attacks [211]. Most adversarial attacks inject crafted high-frequency information to mislead a trained model [191, 67, 153], but such artifacts are non-existent in natural images. Zhao et al. [211] instead proposed a method to find *natural* adversarial examples by performing the perturbation in the latent space of a GAN. While this method finds an example which looks like a specific input, Bayes-TrEx finds high-

confidence misclassifications in the entire data distribution.

4.2.4 Confidence in Neural Networks

Bayes-TrEx can also be used to detect overconfidence in neural networks. An overconfident neural network [71] makes many mistakes with disproportionately high confidence. While many approaches aim to address this network overconfidence problem [20, 58, 121, 194], Bayes-TrEx is complementary to these efforts. Rather than altering the confidence of a neural network, it instead infers examples of a particular confidence. If the model is overconfident, it may return few, if any, samples with ambiguous predictions. At the same time, it may find many misclassifications with high confidence. In our experiments (Sec. 4.4.8), we discover that the popular adversarial discriminative domain adaptation (ADDA) technique produces a more overconfident model than the baseline.

4.3 Methodology

Given a classifier $f : X \to \Delta_K$ which maps a data point to the probability simplex of K classes, the goal is to find an input $\vec{x} \in X$ in a given data distribution $p(\vec{x})$ such that $f(\vec{x}) = \vec{p}$ for some prediction confidence $\vec{p} \in \Delta_K$. We consider the problem of sampling from the posterior

$$p(\vec{x}|f(\vec{x}) = \vec{p}) \propto p(\vec{x})\, p(f(\vec{x}) = \vec{p}|\vec{x}). \tag{4.1}$$

A common approach to posterior sampling is to use Markov Chain Monte-Carlo (MCMC) methods [28]. However, when the measure of the level set $\{\vec{x} : f(\vec{x}) = \vec{p}\}$ is small or even zero, sampling directly from this posterior is infeasible: the posterior being zero everywhere outside of the level set makes it unlikely for a random-walk Metropolis sampler to land on $\vec{x}$ with non-zero posterior [73], and the gradient being zero everywhere outside of the level set means that a Hamiltonian Monte Carlo sampler does not have the necessary gradient guidance toward the level set [149].

To enable efficient sampling, we relax the formulation by "widening" the level set and accepting $\vec{x}$ when $f(\vec{x})$ is close to the target $\vec{p}$ (Fig. 1-1). Specifically, we introduce a random vector $\vec{u} = [u_1, \ldots, u_K]^T$, distributed as

$$u_i|\vec{x} \sim \mathcal{N}\left(f(\vec{x})_i, \sigma^2\right), \tag{4.2}$$

where σ is a hyper-parameter.

Instead of directly sampling from Eqn. 4.1, we can now sample from the new posterior:

$$p(\vec{x}|\vec{u} = \vec{u^*}) \propto p(\vec{x})p(\vec{u} = \vec{u^*}|\vec{x}), \tag{4.3}$$

$$\vec{u^*} = \vec{p}. \tag{4.4}$$

The hyper-parameter σ controls the peakiness of the relaxed posterior. A smaller α makes it closer to the true posterior and makes the distribution peakier and harder to sample, while a larger α makes it closer to the data distribution $p(\vec{x})$ and easier to sample. As σ goes to 0, it approaches the true posterior. Formally,

$$\lim_{\sigma \to 0} p(\vec{x}|\vec{u} = \vec{u^*}) = p(\vec{x}|f(\vec{x}) = \vec{p}). \tag{4.5}$$

While the formulation in Eqn. 4.2 is applicable to arbitrary confidence $\vec{p}$, the dimension of $\vec{u}$ is equal to the number of classes, which poses scalability issues for large numbers of classes. However, for a wide range of interesting use cases of BAYES-TREX, we can use the following reductions:

1. Highly confident in class i: $\vec{p}_i = 1, \vec{p}_{\neg i} = 0$. We have

$$u|\vec{x} \sim \mathcal{N}\left(f(\vec{x})_i, \sigma^2\right), \qquad u^* = 1. \tag{4.6}$$

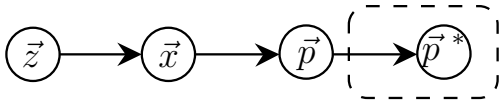


Figure 4-4: The graphical model for the inference problem of finding latent codes $\vec{z}$ which map to images $g(\vec{z}) = \vec{x}$ which exhibit specific prediction confidence $f(\vec{x}) = \vec{p}$. The dashed box indicates the relaxed formulation, where $\vec{p}$ is relaxed to $\vec{p}^{\,*}$.

2. Ambiguous between class i and j: $\vec{p}_i = \vec{p}_j = 0.5$, $\vec{p}_{\neg i,j} = 0$. We have

$$u_1|\vec{x} \sim \mathcal{N}\left(|f(\vec{x})_i - f(\vec{x})_j|, \sigma_1^2\right), \tag{4.7}$$

$$u_2|\vec{x} \sim \mathcal{N}(\min(f(\vec{x})_i, f(\vec{x})_j) - \max_{k \neq i,j} f(\vec{x})_k, \sigma_2^2), \tag{4.8}$$

$$u_1^* = 0, u_2^* = 0.5. \tag{4.9}$$

σ_1 and σ_2 are hyperparameters.

In addition, most high dimensional data distributions, such as those for images, are implicitly defined by a transformation $g : Z \to X$ from a latent distribution $p(\vec{z})$. Consequently, given

$$\vec{x} = g(\vec{z}), \tag{4.10}$$

$$\vec{u}|\vec{z} \sim \mathcal{N}(f(\vec{x}), \sigma^2), \tag{4.11}$$

$$p(\vec{z}|\vec{u} = \vec{u}^*) \propto p(\vec{z})p(\vec{u} = \vec{u}^*|\vec{z}), \tag{4.12}$$

Bayes-TrEx samples $\vec{z}$ according to Eqn. 4.12 and reconstruct the example $\vec{x} = g(\vec{z})$ for model inspection.

4.4 Experiments

4.4.1 Overview

A key strength of Bayes-TrEx is the ability to evaluate a classifier on any data distribution $\mathbb{P}_D$, independent of its training distribution $\mathbb{P}_C$. We demonstrate the versatility of Bayes-TrEx on four relationships between $\mathbb{P}_D$ and $\mathbb{P}_C$ (Fig. 4-5). With

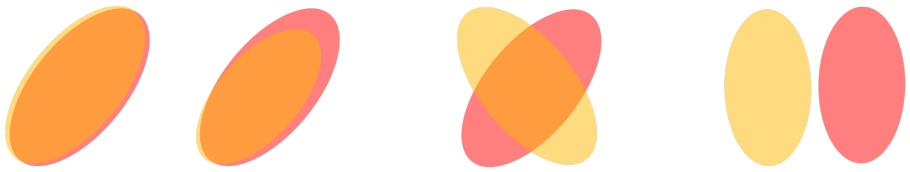
$\mathbb{P}_C = \mathbb{P}_D$ (Fig. 4-5(a)), Sec. 4.4.3 and 4.4.4 present examples that trigger high and ambiguous model confidence and Sec. 4.4.5 presents examples that interpolate between two classes. In Sec. 4.4.6, we consider $\mathbb{P}_D$ with narrower support than $\mathbb{P}_C$ (Fig. 4-5(b)), where the support of $\mathbb{P}_D$ excludes examples from a particular class. In this case, high-confidence examples—as judged by the classifier—correspond to high-confidence misclassifications. In Sec. 4.4.7 and 4.4.8, we analyze the classifier C for novel class extrapolation and domain adaptation behaviors with overlapping or disjoint supports of $\mathbb{P}_C$ and $\mathbb{P}_D$ (Fig. 4-5(c, d)). Representative results are in the main text; further results are in the appendix.

4.4.2 Datasets and Inference Details

We evaluate Bayes-TrEx on rendered images (CLEVR) and organic datasets (MNIST and Fashion-MNIST). For all CLEVR experiments, we use the pre-trained classifier distributed by the original authors[1]. The transition kernel uses a Gaussian proposal for the continuous variables (e.g., x-position) and categorical proposal for the discrete variables (e.g., color), both centered around and peaked at the current value. For (Fashion-)MNIST experiments, architectures and training details are described in Appx. B.1. For domain adaptation analysis, we train ADDA and baseline models using the code provided by the authors[2].

CLEVR images are rendered from scene graphs, on which we define the latent

[1] `https://github.com/facebookresearch/clevr-iep`
[2] `https://github.com/erictzeng/adda`



(a) $\mathbb{P}_D = \mathbb{P}_C$ (b) $\mathbb{P}_D \subsetneq \mathbb{P}_C$ (c) $\mathbb{P}_D \cap \mathbb{P}_C \neq \mathbb{P}_D \neq \mathbb{P}_C$ (d) $\mathbb{P}_D \cap \mathbb{P}_C = \varnothing$

Figure 4-5: Different relations between the classifier training distribution ($\mathbb{P}_C$, red) and Bayes-TrEx data distribution ($\mathbb{P}_D$, yellow). (a) $\mathbb{P}_C$ and $\mathbb{P}_D$ are equal. (b) The support of $\mathbb{P}_D$ is a subset of that of $\mathbb{P}_C$. (c) $\mathbb{P}_D$ and $\mathbb{P}_C$ have overlapping supports. (d) Supports of $\mathbb{P}_C$ and $\mathbb{P}_D$ are disjoint.

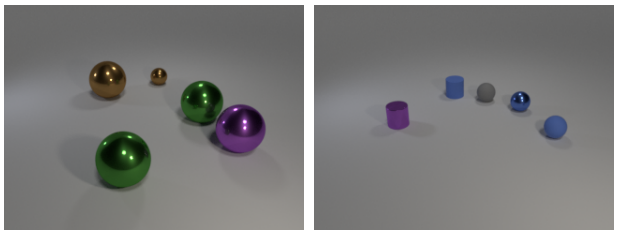
Model	Dataset	FID
VAE	MNIST	72.33
	Fashion-MNIST	87.89
GAN	MNIST	11.83
	Fashion-MNIST	29.44

Table 4.1: Fréchet Inception Distance (FID) for VAE and GAN models trained on the entire dataset. A lower value indicates higher quality. Appx. B.2 presents the statistics for all models.

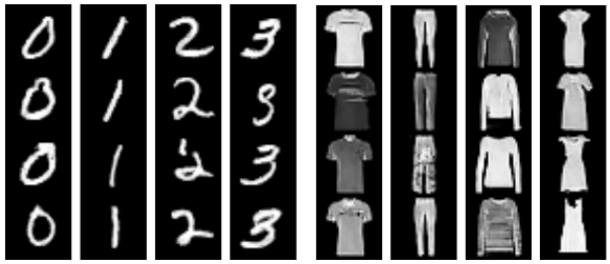
distribution $p(\vec{z})$. Since the (Fashion-)MNIST groundtruth data distribution is unknown, we estimate it using a VAE or GAN with unit Gaussian $p(\vec{z})$. These learned data distribution representations have known limitations, which may affect sample quality [12]. Table 4.1 lists the Fréchet Inception Distance (FID) [79] for two VAE and GAN models, with the full table in Appx. B.2. The FID scores show the GANs generate more representative samples than the VAEs.

We consider two MCMC samplers: random-walk Metropolis (RWM) and Hamiltonian Monte Carlo (HMC). We use the former in CLEVR where the rendering function is non-differentiable, and the latter for (Fashion-)MNIST. For HMC, we use the No-U-Turn sampler [81, 149] implemented in the probabilistic programming language Pyro [18]. We choose $\sigma = 0.05$ for all experiments. Alternatively, σ can be annealed to gradually reduce the relaxation.

Selecting appropriate stopping criteria for MCMC methods is an open problem. State-of-the-art approaches require a gold standard inference algorithm [43] or specific posterior distribution properties, such as log-concavity [68]. As neither of these requirements are met for our domains, we select stopping criteria based on heuristic performance and cost of compute (Appx. B.9). CLEVR requires GPU-intensive rendering, so we stop after 500 samples. (Fashion-)MNIST samples are cheaper to generate, so we stop after 2,000 samples. Empirically, we find each sampling step takes 3.75 seconds for CLEVR, 1.18s for MNIST, and 1.96s for Fashion-MNIST, all on a single NVIDIA GeForce 1080 GPU.



(a) $\vec{p}_{5\text{ Spheres}} = 95.7\%$ (b) $\vec{p}_{2\text{ Blue Sph.}} = 91.1\%$



(c) MNIST (d) Fashion-MNIST

Figure 4-6: High-confidence samples, which pass the smoke test for CLEVR, MNIST, and Fashion-MNIST T-shirt, trousers, pullover, and dress. All of these examples appear to belong to their expected classes and meet the high-confidence behavior target. More examples in Appx. B.3.

4.4.3 High Confidence

As an initial smoke test, we evaluate Bayes-TrEx by finding highly confident examples. (Fashion-)MNIST data distributions are learned by GAN. Fig. 4-6 depicts samples on the three datasets. Additional examples are in Appx. B.3.

4.4.4 Ambiguous Confidence

Next, we find ambiguous (Fashion-)MNIST examples for which the classifier has similar prediction confidence between two classes, using data distributions learned by a VAE. Fig. 4-7 shows ambiguous examples from each pair of classes (e.g. 0v1, 0v2, ..., 8v9). Note the examples presented are ambiguous from the classifier's perspective, though some may be readily classified by a human. Not all pairs result in successful sampling: for example, we were unable to find an ambiguous example with equal prediction confidence between the visually dissimilar classes 0 and 7. These ambiguous examples are useful for visualizing and understanding class boundaries; Appx. B.4 presents a supporting class boundary latent space visualization. *Blended* ambigu-

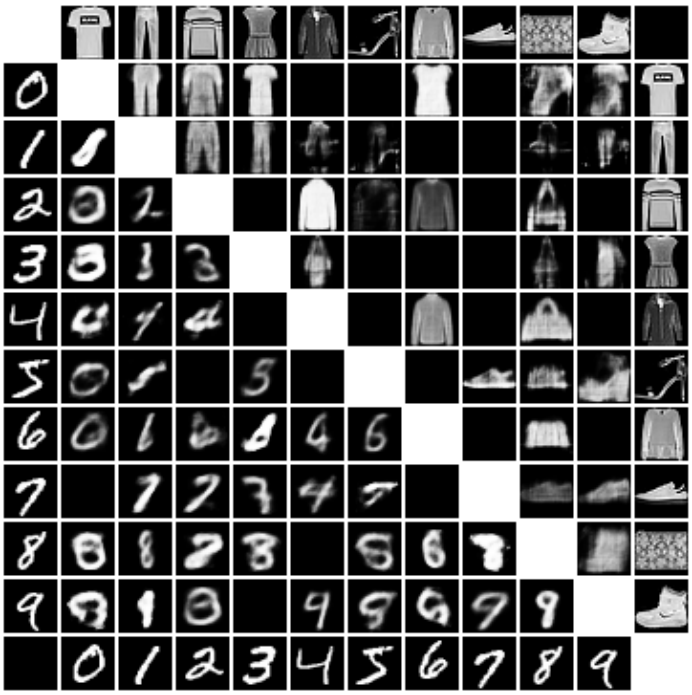


Figure 4-7: Each entry of the matrix is an ambiguous MNIST or Fashion-MNIST example for the classes on its row and column. Blacked-out cells indicate sampling failures. Examples on the outermost edges of the matrix are representative examples from each class (e.g., 0-9 for MNIST).

ous examples have previously been shown to be useful for data augmentation [195]. While these generated ambiguous examples may be similarly useful, we leave this exploration to future work.

Bayes-TrEx can also find examples which are ambiguous across more than two classes; Fig. 4-8 presents samples that are equally ambiguous across all 10 MNIST classes. All these images appear to be very blurry and not very realistic. This is intuitive: even for a human, it would be hard to write a digit in such a way that it is equally unrecognizable across all 10 classes. Details about the sampling formulation and visualizations are presented in Appx. B.4.

In general, for ambiguous examples, we observed only rare successes with data distributions learned by a GAN, which generates sharper and more visually realistic images than a VAE. There are two potential explanations:

1. GAN-distributions prevent efficient MCMC sampling.

2. The classifier rarely makes ambiguous predictions on sharp and realistic images.

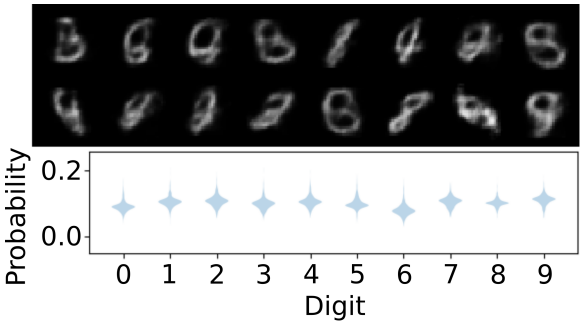


Figure 4-8: Samples of uniformly ambiguous predictions. The violin plot (below) shows that these samples successfully meet this criteria. The samples appear non-descript, though this failure is dissimilar from human failures in classification tasks.

To experimentally evaluate the second explanation, we train a classifier to be consistently ambiguous between class i and $i+1$ for an image of digit i (wrapping around at $10 = 0$) using the following KL-divergence loss:

$$l(y, f(\vec{x})) = \mathbb{KL}(\vec{p}_y, f(\vec{x})), \tag{4.13}$$

$$\vec{p}_{y,i} = \begin{cases} 0.5 & i = y \text{ or } i = (y+1) \bmod 10, \\ 0 & \text{otherwise.} \end{cases} \tag{4.14}$$

Using this classifier, we sample ambiguous examples for 0v1, 1v2, ..., 9v0. Sampling succeeds for all ten pairs, even when using the same GAN model that rarely succeeded in the prior experiment. Fig. 4-9 presents the 0v1 samples and predicted confidence by this modified classifier, and the remaining pairs are visualized in Appx. B.5. Given this sampling success, we conclude that the second explanation is correct.

Bayes-TrEx is also unable to generate ambiguous examples for CLEVR with the manually defined data distribution. Given that the pre-trained classifier only achieves ≈60% accuracy, the result suggests that the model is likely overconfident. Indeed, this has previously been observed in similar settings [99].

4.4.5 Confidence Interpolation

Bayes-TrEx can find examples that interpolate between classes. In Fig. 4-10, we show MNIST samples which interpolate from $(P_8 = 1.0, P_9 = 0.0)$ to $(P_8 = 0.0, P_9 =$

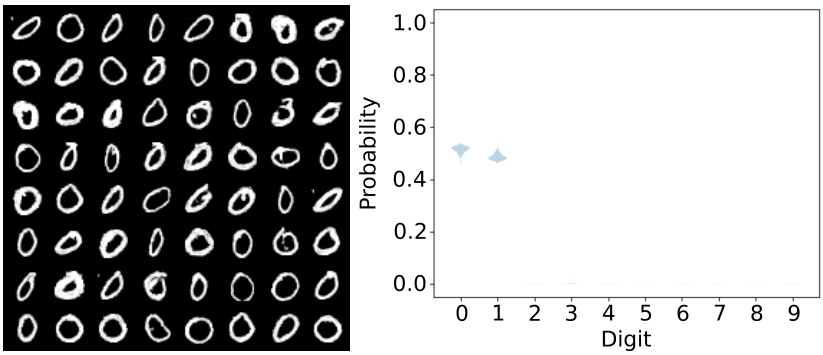


Figure 4-9: Samples which are scored ambiguously between the 0 and 1 classes, and the confidence plot with the GAN distribution and always ambiguous classifier. This shows successful sampling and supports hypothesis 2.

1.0) and Fashion-MNIST samples from ($P_{\text{T-shirt}} = 1.0, P_{\text{Trousers}} = 0.0$) to ($P_{\text{T-shirt}} = 0.0, P_{\text{Trousers}} = 1.0$) over intervals of 0.1, with a VAE-learned data distribution.

The interpolation between two very different classes reveal insights into the model behavior. For example, the interpolation from 8 to 9 generally shrinks the bottom circle toward a stroke, which is the key difference between digits 8 and 9. For Fashion-MNIST, the presence of two legs is important for trousers classification, even appearing in samples with ($\vec{p}_{\text{T-shirt}} = 0.9, \vec{p}_{\text{Trousers}} = 0.1$) (second column). By contrast, a wider top and the appearance of sleeves are important properties for T-shirt classification. These two trends result in most of the interpolated samples having a short sleeve on the top and two distinct legs on the bottom.

4.4.6 High-Confidence Failures

With neural networks being increasingly used for high-stakes decision making, high-confidence failures are one area of concern, as these failures may go unnoticed. BAYES-TREX can find such failures. Specifically, if the data distribution (Fig. 4-5(b)) does *not* include a particular class, then the resulting high-confidence examples correspond to high-confidence *misclassifications* for that class. For example, in Fig. 4-11(a), the CLEVR classifier is highly confident that there is one cube though there is no cube in the image. In App. B.11, the saliency map for Fig. 4-11(a) reveals that classifier mistakes the front shiny red cylinder for a cube. Removing this cylinder causes the

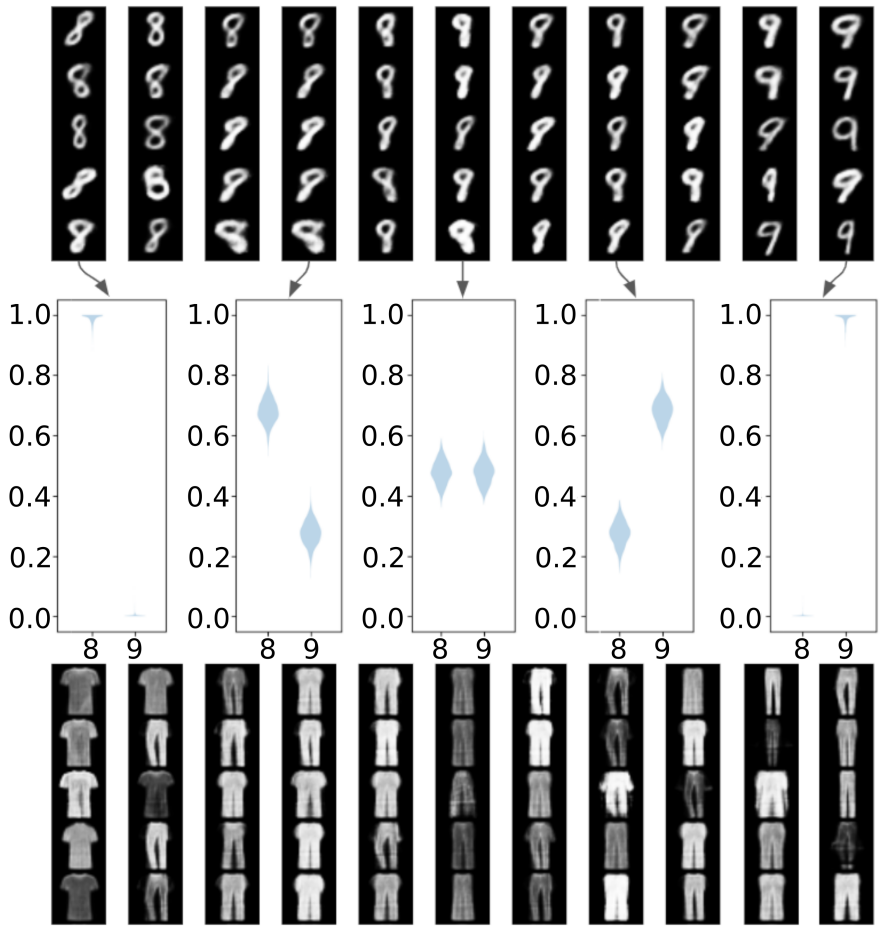
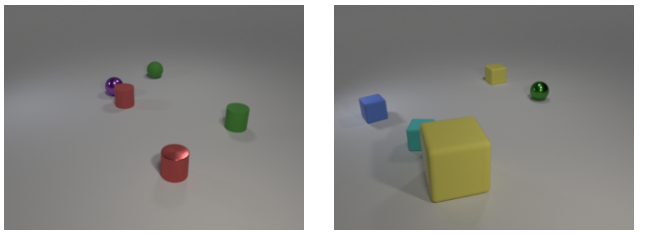


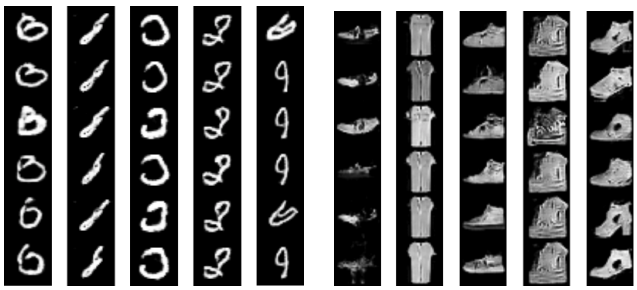
Figure 4-10: Confidence interpolation between digit 8 and 9 for MNIST and between T-shirt and trousers for Fashion-MNIST. Each of the 11 columns show samples of confidence ranging from $[\vec{p}_{\text{class a}} = 1.0, \vec{p}_{\text{class b}} = 0.0]$ (left) to $[\vec{p}_{\text{class a}} = 0.0, \vec{p}_{\text{class b}} = 1.0]$ (right), with an interval of 0.1. Selected confidence plots that demonstrate the sampling successes for MNIST predictions are shown in the middle.

confidence to drop to 29.0%. In addition, such high-confidence failures can also be used for data augmentation to increase network reliability [56].

For (Fashion-)MNIST, a GAN is trained on all data sans a single class, resulting in the learned data distribution excluding the given class. Figs. 4-11(c) and 4-11(d) depict high-confidence misclassifications for digits 0-4 in MNIST and sandal, shirt, sneaker, bag, and ankle boot in Fashion-MNIST, respectively. By evaluating these examples, we can assess how well human-aligned a classifier is. For example, for MNIST, some thin 8s are classified as 1s and particular styles of 6s and 9s are classified as 4s. These results seem intuitive, as a human might make these same mistakes.



(a) $\vec{p}_{1\ \text{Cube}} = 93.5\%$ (b) $\vec{p}_{2\ \text{Cylinders}} = 90.2\%$



(c) MNIST (d) Fashion-MNIST

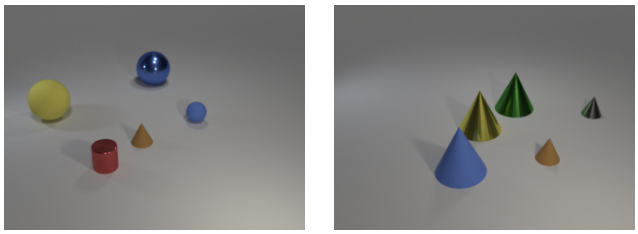
Figure 4-11: High confidence classification failures. (a): CLEVR, 1 Cube. Note that no cube is present in the sample. (b): CLEVR, 2 Cylinders—again, containing no cylinders. (c) MNIST failures for digits 0-4. 0s are composed of 6s; 1s of 8s; 2s of 0s, and so on. (d) Fashion-MNIST failures for sandal, shirt, sneaker, bag, and ankle boot. Additional examples are presented in Appx. B.6.

Likewise, for Fashion-MNIST, most failures come from semantically similar classes, e.g. sneaker $\longleftrightarrow$ ankle boot. Less intuitively, however, chunky shoes are likely to be classified as bags. Additional visualizations are presented in Appx. B.6.

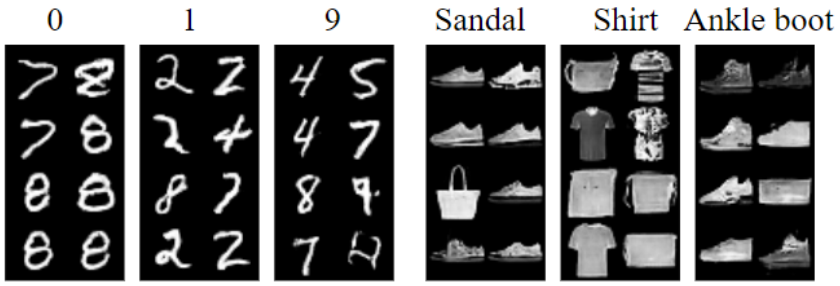
4.4.7 Novel Class Extrapolation

It is important to understand the novel class extrapolation behavior of a model before deployment. During training an autonomous vehicle might learn to safely operate around cyclists and cars. But can we predict how the vehicle will behave when it encounters a novel class, like a tandem bicycle? Bayes-TrEx can be used to understand such behaviors by sampling high-confidence examples with a data distribution that contains novel classes, while excluding the true target classes, Fig. 4-5(c, d).

For CLEVR, we add a novel cone object to the data distribution and remove the existing cube from it. We sample images that the classifier is confident to include cubes, shown in Fig. 4-12 (a, b). A saliency map analysis in Appx. B.11 confirms



(a) $\vec{p}_{1\text{ Cube}} = 98.5\%$ (b) $\vec{p}_{5\text{ Cubes}} = 92.5\%$



(c) MNIST (d) Fashion-MNIST

Figure 4-12: Novel class extrapolation examples. (a, b): For CLEVR, the novel cone objects are mistaken for cubes. (c, d): For (Fashion-)MNIST, we train classifiers on subsets of the data (digits 0, 1, 3, 6, 9 and pullover, dress, sandal, shirt, and ankle boot), and train GANs with the excluded data. Samples for which the classifier is highly confident ($\approx 99\%$) in several target classes are shown (e.g., targets 0, 1, and 9 for MNIST). Additional examples are presented in Appx. B.7.

that the classifier indeed mistakes these cones for cubes. In Appx. B.7, we assess CLEVR's novel class extrapolation for cylinders and spheres, and similarly show the model readily confuses cones for these classes as well.

For MNIST and Fashion-MNIST, we train the respective classifiers on digits 0, 1, 3, 6, 9 and pullover, dress, sandal, shirt and ankle boot classes. We train GANs using only the excluded classes (e.g., digits 2, 4, 5, 7, 8 for MNIST). Using these GANs, we find examples where the classifier has high prediction confidence, as shown in Fig. 4-12 (c, d). For MNIST, there are few reasonable extrapolation behaviors, most likely due to the visual distinctiveness between digits. By comparison, some Fashion-MNIST extrapolations are expected, such as confusing the unseen sneaker class for sandals and ankle boots. However, the classifier also confidently mistakes various styles of bags as sandals, shirts, and ankle boots. App. B.7 contains additional visualizations.

4.4.8 Domain Adaptation

Finally, we use BAYES-TREX to analyze domain adaptation behaviors. We reproduce the SVHN [150] → MNIST experiment studied by Tzeng, et al. [197]. We train two classifiers, a baseline classifier on labeled SVHN data only, and the ADDA classifier on labeled SVHN data and unlabeled MNIST data. Indeed, domain adaptation improves classification accuracy: 61% for the baseline classifier on MNIST vs. 71% for the ADDA classifier.

But is this the whole story? To study model performance in the high-confidence range, we use BAYES-TREX to generate high-confidence examples for both classifiers with the MNIST data distribution learned by GAN, as shown Fig. 4-13. It appears the ADDA model makes *more* mistakes in these images—for example, in the 2nd column in Fig. 4-13(b), all images where the classifier is highly confident to be 1 are actually 0s. To further study this, we hand-label 10 images per class and compute the classifier accuracy on them. Table 4.3 shows the accuracy per digit class, as well as the overall accuracy. This analysis confirms the baseline model is more accurate than the ADDA model on these samples, suggesting that ADDA is more overconfident than the baseline. While this result does not contradict the higher overall accuracy of ADDA, it does caution against deploying such domain adaptation models without further inspection and confidence calibration assessment.

4.4.9 Quantitative Evaluation

We quantitatively evaluate the quality of BAYES-TREX samples by assessing whether the classifier's prediction confidence matches the specified target on the generated examples. Table 4.2 presents the mean and standard deviation of the confidence on a selection of representative settings, and Appx. B.9 lists the full set of such evaluations. The prediction confidences are tightly concentrated around the targets, demonstrating sampler success.

Test	Data	Target	Prediction Confidence
A	M	$\vec{p}_4 = 1$	$1.00 \pm .01$
	F	$\vec{p}_{\text{Coat}} = 1$	$0.98 \pm .02$
	C	$\vec{p}_{2\ \text{Blue Sph.}} = 1$	$0.89 \pm .25$
B	M	$\vec{p}_1 = \vec{p}_7 = 0.5$	$0.49, 0.49 \pm .02, .03$
	F	$\vec{p}_0 = \vec{p}_3 = 0.5$	$0.48, 0.48 \pm .02, .02$
C	M	$\vec{p}_8, \vec{p}_9 = 0.6, 0.4$	$0.58, 0.37 \pm .04, .04$
	F	$\vec{p}_0, \vec{p}_1 = 0.2, 0.8$	$0.17, 0.79 \pm .04, .04$
D	M	$\vec{p}_8 = 1$	$0.98 \pm .02$
	F	$\vec{p}_{\text{Bag}} = 1$	$0.97 \pm .03$
	C	$\vec{p}_{1\ \text{Cube}} = 1$	$0.93 \pm .06$
E	M	$\vec{p}_6 = 1$	$1.00 \pm .01$
	F	$\vec{p}_{\text{Sandal}} = 1$	$1.00 \pm .01$
	C	$\vec{p}_{1\ \text{Cylinder}} = 1$	$0.96 \pm .03$
F	M	$\vec{p}_5 = 1$	$1.00 \pm .01$

Table 4.2: Mean and standard deviation of the sample prediction confidences. Tests are A: high confidence, B: ambiguous, C: interpolation, D: misclassifications, E: novel classes, and F: domain adaptation. Data are M: MNIST, F: Fashion, C: CLEVR. Fashion-MNIST classes 0-9 correspond to T-shirt, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. See Appx. B.9 for full statistics.

	0	1	2	3	4	5	6	7	8	9	All
Base	1	.6	1	.7	.5	.9	.9	.7	1	.7	.8
DA	.9	0	.8	.9	.2	1	.8	1	1	.6	.72

Table 4.3: Per-digit and overall accuracy among high-confidence MNIST samples for the baseline and domain adaptation (DA) models. While DA has higher overall accuracy (0.71 vs. 0.61), it performs *worse* on high-confidence samples (0.72 vs. 0.80). This suggests overconfidence.

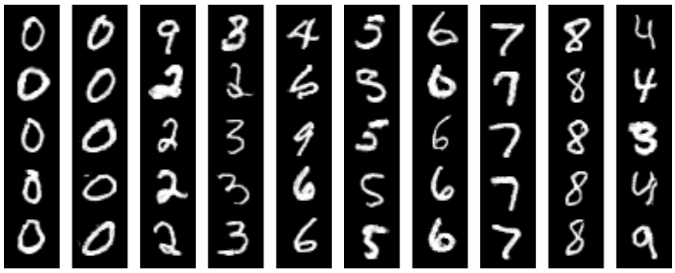
4.4.10 Test-Set Comparison

Standard model evaluations are typically performed on the test set. While inspecting test set examples is not an apples-to-apples comparison for all Bayes-TrEx use cases (e.g. domain adaptation), we study the comparable ones.

Ambiguous Confidence

We find ambiguous examples in the (Fashion-)MNIST datasets where the classifier has confidence in $[40\%, 60\%]$ for two classes. Out of 10,000 test examples on each dataset, we find only 12 MNIST examples across 10 class pairings, and 162 Fashion-MNIST examples across 12 pairings, as shown in Fig. 4-14. By comparison, Bayes-TrEx

(a) Baseline examples



(b) ADDA examples

Figure 4-13: High confidence examples for baseline and ADDA models, classes 0 to 9, showing more misclassifications for the ADDA model. More examples in Appx. B.8.

found ambiguous examples for 38 MNIST pairings and 28 Fashion-MNIST pairings (cf. Fig. 4-7).

High-Confidence Failures

We collect and inspect highly confident test set misclassifications (confidence $\geq 85\%$). For CLEVR, out of $15,000$ test images, the baseline discovers between 0 and 15 examples for each target. Notably, there are no 2-cylinder misclassifications in the test set, but Bayes-TrEx successful generated some (Fig. 4-11(b)).

From the 10,000 test examples in (Fashion-)MNIST, 84 MNIST images and 802 Fashion-MNIST images were confidently misclassified. Upon closer inspection, however, we find that the a large fraction of the failures are actually due to *mislabeling*, rather than misclassification. We manually relabel all 84 MNIST misclassifications and ten Fashion-MNIST misclassifications per class, except for the trousers class which only has 3 misclassifiations. We find that the 60 out of 84 MNIST images 42 out of 93 Fashion-MNIST images are mislabeled, rather than misclassified.

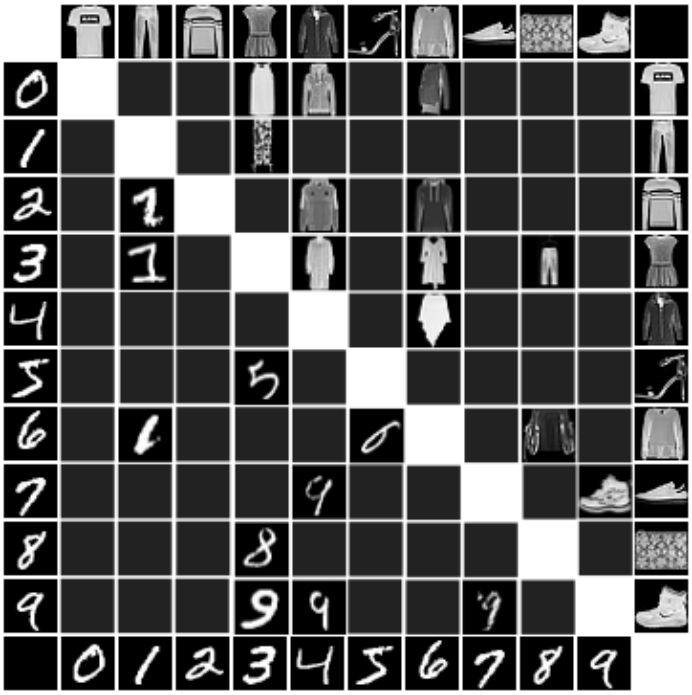


Figure 4-14: Ambiguous examples from the (Fashion-)MNIST test sets. Compared to those found by BAYES-TREX in Fig. 4-7, test set examples have much poorer coverage.

Table 4.4 gives detailed statistics of the number of genuinely misclassified examples. Given the scene graph data representation, all CLEVR misclassifications are genuine. Table 4.5 visualizes some misclassified vs. mislabeled images, with additional classes in Appx. B.10. Identifying mislabeled examples may be useful for correcting the dataset, but is not for our task of model understanding.

CLEVR	Cls.	1 Sph.	1 Cube	1 Cyl.	2 Cyl.
28/28	#	5	8	15	0

MNIST	Cls.	0	1	2	3	4	5	6	7	8	9
24/84	#	3	3	0	5	3	1	3	4	0	2
Fashion	Cls.	0	1	2	3	4	5	6	7	8	9
51/93	#	2	0	9	4	9	1	3	2	1	10

Table 4.4: Number of *genuine* high-confidence misclassifications from test sets. Counts for CLEVR and MNIST are for the entire test set; counts for Fashion-MNIST are for ten random high-confidence misclassifications per class, except for trousers which only has 3 total misclassifications.

Class	Cause	Images
0	Misclass.	
	Mislabeled	
1	Misclass.	
	Mislabeled	
2	Misclass.	∅
	Mislabeled	
Trouser	Misclass.	∅
	Mislabeled	
Bag	Misclass.	
	Mislabeled	

Table 4.5: High confidence misclassifications from the test set. The majority are due to incorrect ground truth labels, not classifier failures. Full table of all classes in Appx. B.10.

Novel Class Extrapolation

In Sec. 4.4.7 analysis, we find that the model mistakes some bags for ankle boots. Interestingly, this propensity is not evident from test set evaluations: the test set confusion matrix in Appx. B.10 shows that no bags are misclassified as ankle boots. This provides further evidence of the value of holistic evaluations with Bayes-TrEx, beyond standard test set evaluations.

4.5 Discussion

Bayes-TrEx is a Bayesian inference approach for generating examples that trigger specified target predictions and so provide insight into model behaviors. These examples can be further analyzed with downstream interpretability methods (Fig. 4-3 and Appx. B.11). To make Bayes-TrEx easier for model designers to use, future work should develop methods to cluster and visualize trends in the generated examples, as well as to estimate coverage of the level set.

For organic data, the underlying data distributions can be learned with VAEs or GANs or other generative model. These have known limitations in sample diver-

sity [12] and are computationally expensive to train, especially for high resolution images. In principle, Bayes-TrEx is agnostic to the distribution learner form and can benefit from future research in this area. In practice, Bayes-TrEx is currently limited to low dimensional latent spaces, as applying MCMC sampling to high dimensional latent spaces is an open problem.

Finally, while we analyzed only classification models with Bayes-TrEx, it also has the potential for analyzing structured prediction models such as machine translation or robotic control. For these domains, dependency among outputs would need to be explicitly taken into account. We plan to extend Bayes-TrEx to these areas in the future.

Chapter 5

Inspect: RL & Robot Controllers

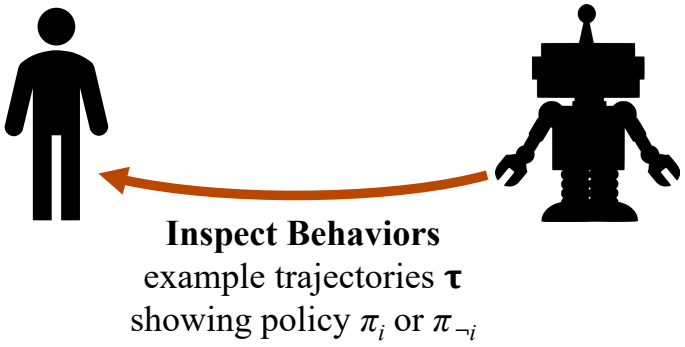


Figure 5-1: This chapter adapts the method introduced in Chapter 4 to inspect robot controllers behaviors instead of classifier behaviors.

This chapter presents an adaptation to the method introduced in Chapter 4 for inspecting classifier behaviors to inspect RL and robot controller behaviors instead. In Chapter 4, the inference goal was to find images $\vec{x}$ such that a classifier f has a prediction confidence score of $\vec{p}$ that is relaxed to make inference feasible. Through exposure to these images, people are better able to inspect the learned behaviors of classifiers. In this chapter, the inference goal is adapted to instead find *environments* or *tasks* such that robot controllers exhibit specific behaviors; this chapter shows that people are better able to inspect the learned behaviors of robot controllers through exposure to these examples. This chapter further demonstrates that this technique is useful for designing specifications by expressly studying how to construct and debug the specification for a dynamical system-based robot controller.

5.1 Introduction

In 2018, after a confluence of failures, an autonomous vehicle (AV) struck and killed a pedestrian for the first time. In the run-up to this fateful event, the responsible company had reportedly been trying to improve the AV "ride experience" by emphasizing non-critical behaviors—such as the smoothness of the ride [26]. This event reflects the long-standing challenge in robotics: designing an appropriate objective which considers both safety-critical and non-critical behaviors. When crafting an objective, it is virtually impossible to proactively account for all potential controller behaviors, and some priorities may even be in conflict with one another [168]. In practice, any given robot behaviors may be specified, unspecified, or even misspecified [21], so extensive testing and evaluation is a critical component of designing and assessing robot controllers—especially those using black-box models such as deep neural networks.

A common testing procedure focuses on finding extreme and edge cases of controller failure. For example, a tester might use this procedure to find that the AV swerves very badly when encountering a farm animal while traveling at 60mph. Finding such extreme and edge cases is well-studied within both traditional software testing paradigms [147] and more recent adversarial perturbation testing methods [67]. However, we argue that an equally, if not more, important form of testing should focus on *representative* scenarios, which considers the likelihood of encountering these scenarios. For example, if this AV is going to be deployed exclusively in New York City, the above example is largely unhelpful: cars rarely travel at 60mph in the city, and are very unlikely to encounter farm animals. Instead, the tester may prefer to know that the car swerves—though not as substantively—at lower speeds when a pedestrian steps toward it. Finding representative scenarios is often overlooked, but is especially useful for robotics. This is the focus of this paper.

The content of this chapter largely reproduces the text from Zhou, Booth, Figueroa, and Shah's CoRL 2021 paper: *RoCUS: Robot controller understanding via sampling* [213].

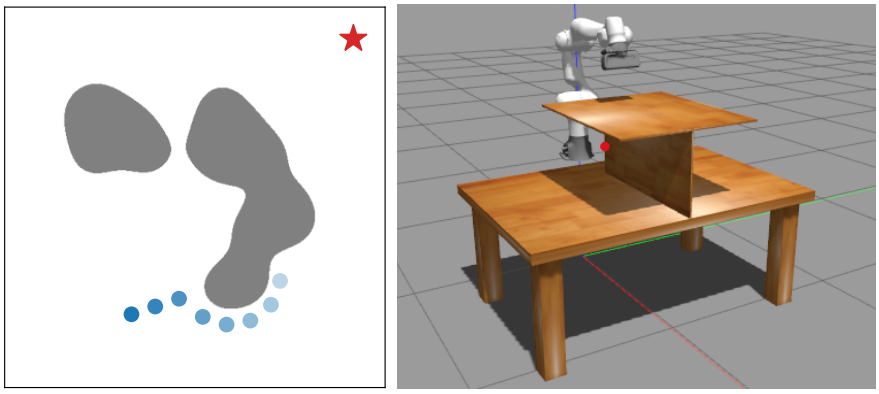


Figure 5-2: Two use case demonstrations of RoCUS: 2D navigation (left) and 7DoF arm reaching (right). In the 2D navigation environment, the robot must navigate from the bottom left corner to the top right corner; the position, count, and shapes of the obstacles vary. In the 7DoF arm reaching environment, the robot must reach a target (the red dot). The position of this target varies, and is either positioned under the left or right sides of the T-shaped divider.

Explicit mathematical analysis of robot controllers is implausible given the high dimensionality of the configuration space and the potential black-box representation of a learned controller. With access to an environment simulator, though, a straightforward testing approach is to roll out the robotic controller on various environments (e.g. road conditions under different weather and congestion, with or without farm animals or pedestrians, etc.), and analyze those rollouts that exhibit a specified behavior—like excessive swerving. However, with too few environments, we risk missing the condition(s) that triggers the target behavior most saliently. With too many environments, all the most salient rollouts would be close to the global maximum at the expense of diversity and coverage. For example, if a farm animal causes the most swerving, followed by a pedestrian and a dangling tree branch, using too few environments may only find the pedestrian and the tree branch while using too many would result in an exclusive focus on the farm animal. Neither case helps the human develop a correct mental model of the AV's behavior.

To address this, we introduce Robot Controller Understanding via Sampling (RoCUS), a method to enable systematic behavior inspection. RoCUS finds scenarios that are both inherently likely and elicit specified behaviors by formulating the problem as one of Bayesian posterior inference. Analyzing these scenarios and the resulting tra-

jectories can help developers better understand the robot behaviors, and allow them to iterate on algorithm development if undesirable ones are revealed.

We use RoCUS to analyze three controllers on two common robotics tasks (Fig. 5-2). For a 2D navigation problem, we consider imitation learning (IL) [11], dynamical system (DS) [86], and rapidly-exploring random tree (RRT) [118]. For a 7DoF arm reaching problem, we consider reinforcement learning (RL) [190], as well as the same DS and RRT controllers. For each problem and controller, we specify several behaviors and visualize representative scenarios and trajectories that elicit those behaviors. Through this analysis, we uncover insights that would be hard to derive analytically and thus complement our mathematical understanding of the controllers. Moreover, we include a case study on how to improve a controller based on new insights from RoCUS. As such, RoCUS is a step towards the broader goal of building more accurate human mental models and enabling holistic evaluation of robot behaviors.

5.2 Related Work

RoCUS sits at the intersection of efforts to understand complex model behaviors and those to benchmark robot performance. Methods to understand, interpret, and explain model behaviors are commonplace in the machine learning community. Mitchell et al. introduced Model Cards, a model analysis mechanism which breaks down model performance for data subsets [141]. Model cards have been widely adopted and adapted for new settings, like reinforcement learning [65]. In natural language processing, Ribeiro et al. introduced a checklist for the evaluation of model capabilities and test case generation [173]. In robotics, Fan et al. introduced a verification framework for assessing machine behavior by sampling parameter spaces to find temporal logic-satisfying behaviors [51]. Other efforts aim to summarize robot policies, trading off factors like brevity, diversity and completeness [76, 114]. All of these works have a shared underlying theme: treating the black box as immutable and performing downstream analyses of machine behavior [167]. RoCUS shares this theme and searches for instances which exhibit target behaviors to inform conceptual model formation.

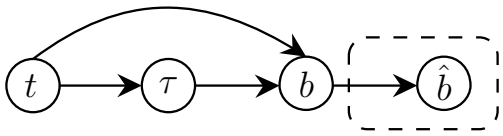


Figure 5-3: The graphical model for the inference problem of finding tasks t and trajectories τ which exhibit specific behaviors b. The dashed box indicates the relaxed formulation (Eq. 5.2). Note the similarities to Figure 4-4 in Chapter 4 from the classification setting.

While the need for benchmarking robot performance is often expressed [134, 146, 91], these efforts usually operate on distributions of trajectories or randomly selected trajectories, and the accompanying metrics are typically task-completion based without consideration of implicit performance factors. Anderson et al. put forth a recommendation of using *success weighted by path length* for navigation tasks—a task-completion metric [8]. Cohen et al. [40] and Moll et al. [145] introduced suites of metrics for comparing motion planning approaches, and Lagriffoul et al. [115] presented a set of task and motion planning scenarios and metrics. Again, all of these proposed metrics are based solely on task completion. Lemme et al. [123] proposed a set of performance measures for reaching tasks, which are either task-completion based or require a costly human motion ground truth. Our contribution is distinct in two ways. First, we propose to sample specific trajectories which communicate controller behaviors instead of reporting metrics averaged over distributions of trajectories. Second, we introduce metrics which draw on these prior works while also including essential alternative and typically emergent quality factors, like motion jerkiness and legibility [47].

5.3 RoCUS

At a high level, RoCUS helps users understand robotic controllers via representative scenarios that exhibit various specified behaviors. It solves this by directly incorporating the distribution of scenarios, formally called *tasks*, into a Bayesian inference framework as shown in Fig. 5-3. A robotic problem is represented by a distribution

$\pi(t)$ of individual tasks t. For example, a navigation problem may have $\pi(t)$ representing the distribution over target locations and obstacle configurations. Given a specific task t, the controller under study induces a distribution $p(\tau|t)$ of possible trajectories τ. If both the controller and the transition dynamics are deterministic, $p(\tau|t)$ reduces to a δ-function at the induced trajectory τ. Stochasticity in either the controller (e.g., RRT) or the dynamics (e.g., uncertain outcome from an action) can result in τ being random. Finally, a behavior function $b(\tau, t)$ computes the behavior value of the trajectory—for example, the motion jerkiness. Some behaviors only depend on the trajectory and not the task, but we use $b(\tau, t)$ for consistency. Sec. 5.4 presents a list of behaviors.

The discussion on behavior in Sec. 5.1 is informal and implicitly combines two related but different concepts. The first concept is the behavior function $b(\tau, t)$ discussed above. The second is the specified target: for the swerving example, we are particularly interested in *maximal* behavior values. Thus, the target value can be thought of as $+\infty$. This inference problem uses the *maximal* mode of RoCUS. In other cases, we are also interested in tasks and trajectories whose behaviors *matches* a target. For example, we want to find road conditions that lead to a daily commute time of an hour, where the behavior is the travel time. This inference problem uses the *matching* mode. Since matching mode is conceptually simpler, we present it first, followed by maximal mode. The sampling procedure is the same for both modes and is presented last in Algorithm 1.

5.3.1 Matching Mode

The objective is to find tasks and trajectories that exhibit user-specified behaviors b^*:

$$t, \tau \sim p(t, \tau | b = b^*) \propto p(b = b^* | t, \tau) p(\tau | t) \pi(t). \tag{5.1}$$

In most cases this posterior does not admit direct sampling, and an envelope distribution is not available for rejection sampling. Markov-Chain Monte-Carlo (MCMC) sampling does not work either: since the posterior is only non-zero on a very small or

even measure-zero set, a Metropolis-Hastings (MH) sampler [73] can get stuck in the zero-density region. Similar to the BAYES-TREX formulation [25], we relax it using a normal distribution formulation as shown in Fig. 5-3:

$$\widehat{b}|b \sim \mathcal{N}(b, \sigma^2) \quad t, \tau \sim p(t, \tau|\widehat{b} = b^*) \propto p(\widehat{b} = b^*|t, \tau)p(\tau|t)\pi(t). \tag{5.2}$$

This relaxed posterior is non-zero everywhere $\pi(t)$ is non-zero and provides useful guidance to an MH sampler. While σ is a hyper-parameter in BAYES-TREX [25], we instead choose σ such that

$$\int_{b^*-\sqrt{3}\sigma}^{b^*+\sqrt{3}\sigma} p(b)\, db = \alpha, \quad \text{with} \quad p(b) = \int_t \int_\tau p(\tau|t)\pi(t)\mathbb{1}_{b(\tau,t)=b}\, d\tau\, dt \tag{5.3}$$

being the marginal distribution of $b(\tau, t)$, which can be estimated by trajectory roll-outs. This formulation has two desirable properties. First, it is scale-invariant with respect to $b(\tau, t)$ such that this factor can be measured with different units like meters or centimeters. Second, the hyper-parameter $\alpha \in [0, 1]$ has the intuitive interpretation of the approximate "volume" of posterior samples $t, \tau \mid \widehat{b} = b^*$ under the marginal $p(t, \tau) = p(\tau|t)\pi(t)$, a notion of their representativeness. The details of this derivation are described in Appendix C.1.

5.3.2 Maximal Mode

In maximal mode, ROCUS finds trajectories that lead to maximal behavior values: $b^* \to \pm\infty$. It can also be used for finding minimal behavior values by negating the behavior. The posterior formulation is:

$$b_0 = \frac{b - \mathbb{E}[b]}{\sqrt{\mathbb{V}[b]}}, \quad \beta = \frac{1}{1 + e^{-b_0}}, \quad \widehat{\beta} \sim \mathcal{N}\left(\beta, \sigma^2\right), \quad t, \tau \sim p(t, \tau|\widehat{\beta} = 1), \tag{5.4}$$

where $\mathbb{E}[b]$ and $\mathbb{V}[b]$ are the mean and variance of the marginal $p(b)$.

Algorithm 1: MH Sampling Procedure

Input: "Posterior volume" α, number of samples N, optional burn-in N_B and thinning period N_T.

1 samples $\leftarrow$ [ ];
2 Get σ from α by Eq. 5.3 (matching) or 5.5 (maximal);
3 Randomly initialize t;
4 **for** $i = 1, ..., N$ **do**
5 $\quad t_{\text{new}}, p_{\text{for}}, p_{\text{rev}} = \text{propose}(t)$
6 $\quad$ Get p from t by Eq. 5.2 (match) or Eq. 5.4 (max)
7 $\quad$ Get p_{new} from t_{new} by Eq. 5.2 or Eq. 5.4;
8 $\quad a \leftarrow (p_{\text{new}} \cdot p_{\text{rev}})/(p \cdot p_{\text{for}})$;
9 $\quad$ Sample $u \sim \mathcal{U}[0, 1]$;
10 $\quad$ **if** $u < a$ **then**
11 $\quad\quad t \leftarrow t_{\text{new}}$;
12 $\quad$ Append t to samples;
13 Optionally, discard the first N_B burn-in samples and thin the samples by only keeping every N_T samples;
14 **return** samples

σ is chosen such that:

$$\int_{1-\sqrt{3}\sigma}^{1} p(\beta)d\beta = \alpha, \tag{5.5}$$

where $p(\beta)$ is the marginal distribution similar to Eq. 5.3. If $p(b)$ is normal, $p(\beta)$ is logit-normal. This formulation is again scale-invariant and has the same "volume" interpretation for α (Appendix C.1).

5.3.3 Posterior Sampling

The posterior sampling mechanism depends on the stochasticity of both the controller and the environment dynamics.

Deterministic Controller & Dynamics: When both the controller and the dynamics are deterministic, so is $\tau|t$, denoted as $\tau(t)$. Eq. 5.2 reduces to $t \sim p(t|\widehat{b} = b^*) \propto p(\widehat{b} = b^*|t, \tau(t))\pi(t)$, and similarly for Eq. 5.4.

Alg. 1 presents the MH sampling procedure. First, σ is computed from α (Line 2). Then we start with an initial task t (Line 3). For each of the N iterations, we propose

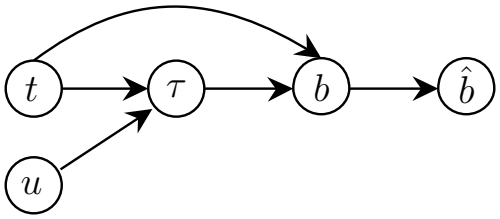


Figure 5-4: The same graphical model as in Fig. 5-3, but with the addition of stochasticity u in the controller such that $\tau|t, u$ is now deterministic.

a new task t_{new} according to a transition kernel and compute the forward and reverse transition probabilities $p_{\text{for}}, p_{\text{rev}}$ (Line 5). We evaluate the posteriors under t and t_{new} (Line 6 and 7) and calculate the acceptance probability using the MH detailed balance principle (Line 8). Finally, we accept or reject accordingly (Line 9 – 11). Note that if the proposal is rejected, the current t is left unchanged *and appended to the samples.* We can discard the first N_B samples as burn-in, and/or thin the samples by a factor of N_T to reduce auto-correlation.

Stochastic Controller: When the controller and $p(\tau|t)$ are stochastic, the controller can usually be implemented by sampling a random variable u (independent from t), and then producing the action based on the realization of u, as shown in Fig. 5-4. For instance, a Normal stochastic policy $\pi(s) \sim \mathcal{N}(\mu(s), \sigma(s)^2)$ can be implemented by first sampling $u \sim \mathcal{N}(0, 1)$ and then computing $\pi(s) = \mu(s) + u \cdot \sigma(s)$. In this case, we sample in the combined (t, τ)-space, with Eq. 5.2 being $p(t, \tau|\widehat{b} = b^*) \propto p(\widehat{b} = b^*|t, \tau(t, u))p(u)\pi(t)$, where we overload $\tau(t, u)$ to refer to the *deterministic* trajectory given the task t and controller randomness u. It is crucial that for any u, we can evaluate $p(u)$. Concretely, modifying Algorithm 1, u_{new} is proposed alongside with t_{new} (Line 5), the detailed balancing factor (Line 8) is multiplied by $p_{u,\text{rev}}/p_{u,\text{for}}$, and $t_{\text{new}}, u_{\text{new}}$ are accepted or rejected together (Line 10 – 12).

Stochastic Dynamics: Using the same logic, RoCUS can also accommodate dynamics stochasticity, *as long as it can be captured in a random variable v and $p(v)$ can be evaluated.* We leave the details to Appendix C.2 and use deterministic dynamics in our experiments.

5.3.4 The Bayesian Posterior Sampling Interpretation

RoCUS uses Bayesian sampling concepts of prior, likelihood, and posterior quite liberally. Specifically, the task distribution is defined as the prior, and thus the notion of a task being likely in the deployment context refers to high probability under the prior. Likelihood refers to the behavior saliency: how much the exhibited behavior matches the behavior specification. The act of posterior sampling then finds tasks that strike a balance between these two objectives. The choice of explicitly modeling the task distribution is intentional, as it is not unlikely that the deployment environment will be different than the development environment. Such a domain mismatch may cause catastrophic failures, especially for learned controllers whose extrapolation behaviors are typically undefined. With a suitable task distribution, RoCUS allows more failures to surface during this testing procedure.

5.4 Behavior Taxonomy

Robot behaviors broadly belong to one of two classes: intentional and emergent. *Intentional* behaviors are those that the controller explicitly optimize with objective functions. For example, the controller for a reaching task likely optimizes to move the end-effector to the target, by setting the target as an attractor in DS, using a target-reaching objective configuration in RRT, or rewarding proximity in RL. Thus, the final distance between the end-effector and the target is an intentional behavior for all three controllers. By contrast, *emergent* behaviors are not explicitly specified in the objective. For the same reaching problem, an RL policy with reward based solely on distance may exhibit smooth trajectories for some target locations and jerky ones for others. Such behaviors may emerge due to robot kinematic structure, training stochasticity, or model inductive bias.

For trajectory τ, many behavior metrics $b(\tau, t)$ can be expressed as a line integral $\int_\tau V(\vec{x})ds$ of a scalar field $V(\vec{x})$ along τ or its length-normalized version $\frac{1}{||\tau||}\int_\tau V(\vec{x})ds$, where ds is the infinitesimal segment on τ at $\vec{x}$ and $||\tau||$ is the trajectory length. $\vec{x}$ and τ can be in either joint space or task space. We introduce six behaviors: length, time

derivatives (velocity, acceleration and jerk), straight-line deviation, obstacle clearance, near-obstacle velocity and motion legibility, whose mathematical expressions are in Appendix C.3. In addition, custom behaviors can also be used with RoCUS.

5.5 RoCUS Use Case Demos

In this section, we demonstrate how RoCUS can find "hidden" properties of various controllers for two common tasks, navigation and reaching. We also uncover a suboptimal controller design due to bad hyper-parameter choices, which is improved based on RoCUS insights.

5.5.1 Controller Algorithms

We consider four classes of robot controllers. The **imitation learning** (IL) controller uses expert demonstrations to learn a neural network policy which maps observations to deterministic actions. The **reinforcement learning** (RL) controller implements proximal policy gradient (PPO) [180]. While a mean and a variance is used to parameterize a PPO policy during training, the policy deterministically outputs the mean action during evaluation. The **dynamical system** (DS) controller modulates the linear controller $\vec{u}(\vec{x}) = \vec{x}^* - \vec{x}$, for the task-space target $\vec{x}^*$, into $\vec{u}_M(\vec{x}) = M \cdot \vec{u}(\vec{x})$ using the modulation matrix M derived from obstacle configuration, as proposed by [86]. We give a self-contained review in Appendix C.4. The **rapidly-exploring random tree** (RRT) controller finds a configuration-space trajectory via RRT and then controls the robot through descretized segments. Notably, RRT is stochastic, and we discuss the use of controller stochasticity u (c.f. Fig. 5-4) in Appendix C.5. The MCMC sampling uses a Gaussian drift kernel, as detailed in Appendix C.6.

5.5.2 2D Navigation Task Experiments

Setup In a rectangular arena with irregularly shaped obstacles, a point mass robot needs to move from the lower left to the upper right corner (Fig. 5-2 left). Ap-

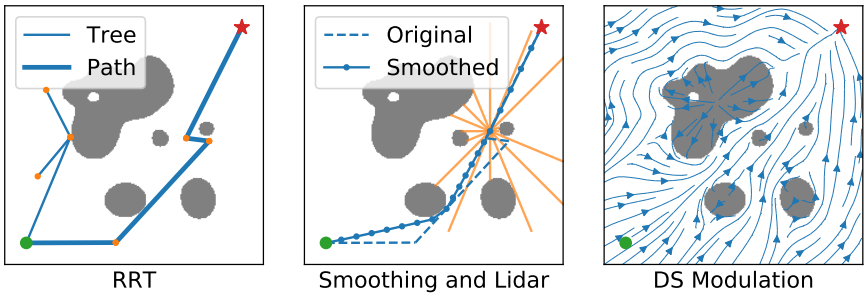


Figure 5-5: RRT, IL and DS controllers on 2D navigation domain. Left: the RRT controller tree. Middle: smoothed RRT trajectory and lidar sensor (orange lines) for IL controller training. Right: the modulation by the DS controller.

pendix C.7 details the obstacle generation and robot simulation procedures and contains more environment visualizations.

We consider three controllers for this environment: an RRT planner, a deep learning IL policy, and a DS (Fig. 5-5). The RRT planner implements Algorithm 2 and discretizes the path to small segments as control signals at each time step. The IL controller uses smoothed RRT trajectories as expert demonstrations, and learns to predict heading angle from its current position and lidar readings. The DS controller finds an interior reference point for each obstacle, and converts each obstacle in the environment to be star-shaped. Γ-functions are then defined for these obstacles and used to compute the modulation matrix M. Appendix C.8 contains additional implementation details.

Straight-Line Deviation In most cases, the robot cannot navigate directly to the target in a straight line. Thus, the collision-avoidance behavior is a crucial aspect for navigation robots. To understand it, we sample obstacles that lead to trajectories minimally deviating from the straight line path. Since the deviation is always non-negative, we use the matching mode in Equation 5.2 with target $b^* = 0$.

In Fig. 5-6, the top row plots posterior trajectories in orange, with prior trajectories in blue. The bottom row plots the obstacle distributions compared to the prior, with red regions being more likely to be occupied by obstacles and blue ones less likely to be obstructed.

For DS and RRT, the posterior trajectories and obstacle configurations are mostly

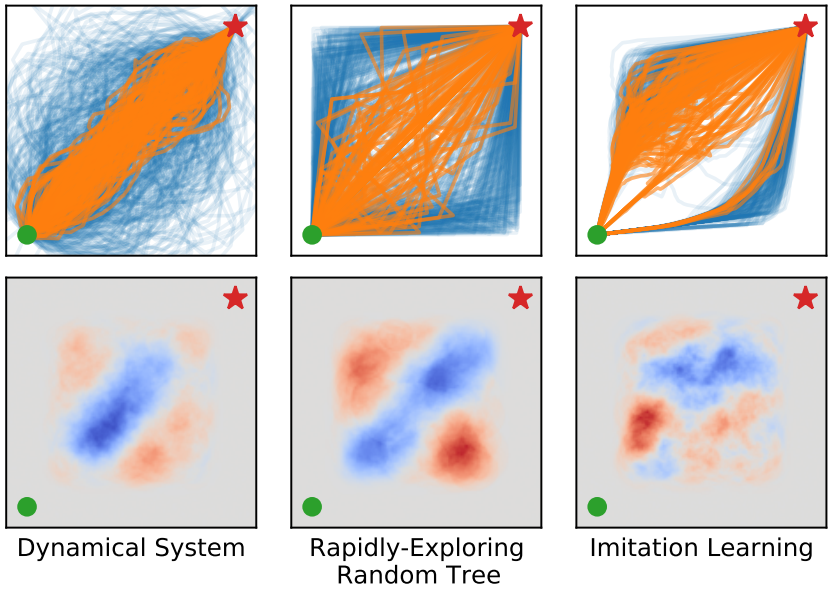


Figure 5-6: Top: Posterior trajectories in orange vs. prior in blue for minimal straight-line deviation behavior for three controllers. Bottom: Posterior obstacle distribution relative to the prior. Higher obstacle density regions are painted in red and lower ones in blue.

symmetric with respect to the straight-line connection, as expected since both methods are formulated symmetrically with respect to the x- and y-coordinates. The obstacle distribution under RRT is also expected, since it seeks straight-line connections whenever possible and thus favor a "diagonal corridor" with obstacles on either side. For DS, however, obstacles are slightly *more* likely to exist at the two ends of the above-mentioned corridor. This behavior is an artifact of the DS *tail effect*, which drags the robot around the obstacle (details in Appendix C.4). By taking advantage of anchor-like obstacles at the ends of the corridor, the modulation can reliably minimize the straight-line deviation.

By comparison, the IL controller saliently exhibits trajectory asymmetry: it mostly takes paths on the left. It is possible that the asymmetry is due to "unlucky" samples by the MH sampler, but many independent restarts all confirm its presence, indicating that the asymmetry is inherent in the learned model. Since the neural network architecture is symmetric, we conclude that the stochasticity in the dataset generation and training procedure (e.g. initialization) leads to such imbalanced behaviors.

Furthermore, the obstacle map suggests that obstacles are distributed very close to the robot path. Why does the robot seem to drive into obstacles? The answer lies in dataset generation: the smoothing procedure (Fig. 5-5 middle) results in most demonstrated paths navigating tightly around obstacles, and it is thus expected that the learned IL controller displays the same behavior.

Takeaways RoCUS reveals two unexpected phenomena. First, IL trajectories are asymmetric toward the left of the obstacle due to dataset and the training imbalance. Second, both DS and IL models exhibit certain "obstacle-seeking" behaviors, the former due to the "tail-effect" and the latter due the dataset generation process. In both cases, such behavior may be undesirable in deployment due to possibly imprecise actuation, and the controller design may need to be modified. Additional studies on legibility and obstacle clearance behaviors are presented in Appendix C.9.

5.5.3 7DoF Arm Reaching Task Experiments

Setup A 7DoF Franka Panda arm is mounted on the side of a table with a T-shaped divider (Figure 5-2 right). Starting from the same initial configuration on top of the table, it needs to reach a random location on either side under the divider. We simulate this task in PyBullet [41]. We consider three controllers: an RRT planner, a deep RL PPO agent, and a DS formulation.

RRT again implements Algorithm 2, but uses inverse kinematics (IK) to first find the joint configuration corresponding to the target location. The RL controller is a multi-layer perceptron (MLP) network trained using the PPO algorithm. The DS model outputs the end-effector trajectory in the task space, which is converted to joint space via IK, with SVM-learned obstacle definitions. Appendix C.10 contains additional implementation details for each method. Overall, RRT and RL are quite successful in reaching the target while the DS is not due to the bulky robot structure, close proximity to the divider, and the task-space only modulation.

End-Effector Movement We find configurations that minimize the total travel distance of the end-effector for RRT and RL (DS omitted due to high failure rate). Figure 5-7 (left two) shows the posterior target locations and trajectories. Notably,

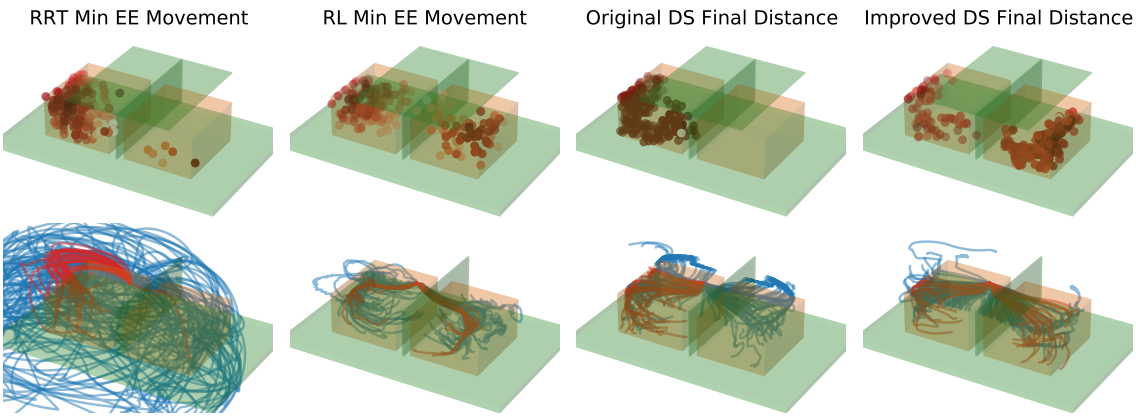


Figure 5-7: Left: Minimal end-effector movement samples for RRT and RL. Right: Posterior samples for minimal distance from end-effector to target for the original and improved DS controllers. Top: posterior targets locations, with tabletop + divider in green and target region in orange. Bottom: posterior trajectories in red, prior trajectories in blue. Robot is mounted on the near long edge.

unlike RL, RRT trajectories are highly asymmetric, since there are straight-line connections in the configuration space from the initial pose to some target regions on the left, while every right-side goal requires at least an intermediate node.

DS Improvement with RoCUS Our initial DS implementation frequently fails to reach the target. This is understandable, as the DS convergence guarantee [86] is only valid in task space, in which the modulation is defined. When the full-arm motion is solved via IK, it is possible that some body parts may collide and get stuck because of the table divider. To understand the DS behaviors, we use RoCUS to sample target locations that result in minimal final distance from the end-effector to the target (i.e., most successful executions, Figure 5-7 center-right). Similar to the RRT case, the samples show strong lateral asymmetry, with all posterior target locations on the left, due to the same cause of asymmetric kinematic structure. The result points to a clear path to improve the DS controller such that it can succeed with right-side targets: increase the collision clearance of the divider so that the end-effector navigates farther away from the divider, thus also bringing the whole arm to be farther away. As detailed in Appendix C.11, this modification greatly improves the controller performance as confirmed by the new symmetry in Figure 5-7 (rightmost). In addition, since the issue with DS controller mainly lies in obstacle avoidance in

joint-space or on the body of the robot, additional techniques [98, 169, 140, 199] could be used and we leave them to future directions.

Takeaway The set of studies reveal an important implication of the robot's kinematic structure: the left side is much less "congested" with obstacles than the right side in the configuration space. While the RL controller is able to learn efficient policies for both sides, the design of certain controllers may need to explicitly consider such factors. AppendixC.11 includes an additional study on legibility.

5.5.4 Quantitative Summary

We studied other additional behaviors on both tasks, and Tab. 5.1 summarizes prior vs. posterior mean behavior values and shows that RoCUS consistently finds samples salient in the target behavior.

Domain	Behavior	Target	Prior (DS)	Post. (DS)	Prior (IL/RL)	Post. (IL/RL)	Prior (RRT)	Post. (RRT)
2D Nav	Avg. Jerk	0	1.84e-3	1.46e-3	6.95e-4	3.19e-4	4.24e-4	2.79e-4
	Straight	0	0.256	0.084	0.378	0.301	0.470	0.162
	Legibility	min	0.819	0.650	0.877	0.784	0.798	0.669
	Obstacle	0	0.309	0.229	0.262	0.218	0.312	0.241
	Obstacle	max	0.309	0.611	0.262	0.387	0.312	0.442
Arm	Straight	0	0.980	0.913	0.858	0.762	1.223	0.897
	EE Dist	0	0.934	0.623	0.958	0.691	3.741	1.192

Table 5.1: Quantitative results on additional behavioral targets for the two domains.

5.6 MCMC Sampling Evaluation

After confirming that RoCUS can indeed uncover significant and actionable controller insights, we turn our attention to evaluating the sampling procedure itself using the tasks described above as motivating examples.

Mixing Property One potential downside of using the MCMC sampler is the slow mixing time, which causes the chain to take a long time to converge from initialization and causes consecutive samples to be highly correlated. We ask: does this phenomenon present in our sampling efforts with RoCUS? Figure 5-8 plots the behavior along the MCMC iterations for the DS minimal straight-line deviation

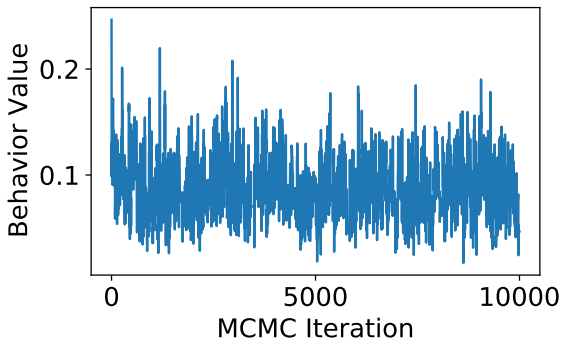


Figure 5-8: 2D navigation DS minimized straight-line deviation samples.

behavior, showing that the chain mixes well quite fast (additional ones in Figure C-2 of Appendix C.6). Thus, a modest amount of samples, such as several thousand, is typically sufficient to model the target posterior distribution well.

Baseline: Top-k Selection To the best of our knowledge, RoCUS is the first work that applies the transparency-by-example formulation [25] to robotic tasks, and we are not aware of existing methods for the same purpose. Notably, adversarial perturbation algorithms [67] are *not* feasible, since stepping in simulator (or real world) is not typically differentiable. Section 5.1 discusses a straightforward alternative that runs the controller on N different scenarios and pick the top-k with respect to the target behavior. We demonstrate its shortcomings on the minimal straight-line deviation behavior for the 2D navigation DS controller. Specifically, RoCUS samples are shown in Figure 5-6, left.

Figure 5-9 (left) shows the trajectories of different values of k for the same fixed N, and vice versa. While a bigger N/k ratio leads to more salient behaviors in the top-k samples, these examples become more concentrated around the global maximum and less diverse, making this approach especially myopic. Further, it is not easy to find the optimal N to trade off between diversity and saliency of the top-k samples. By contrast, RoCUS offers the intuitive α hyper-parameter. Figure 5-9 (middle) shows that a smaller N fails to highlight the "corridor" pattern while a larger N makes it completely open and misses the "tail-effect anchors" at the two ends.

In addition, the hard cut-off at the k-th salient behavior threshold has two undesirable implications: first, every trajectory more salient than the threshold is kept

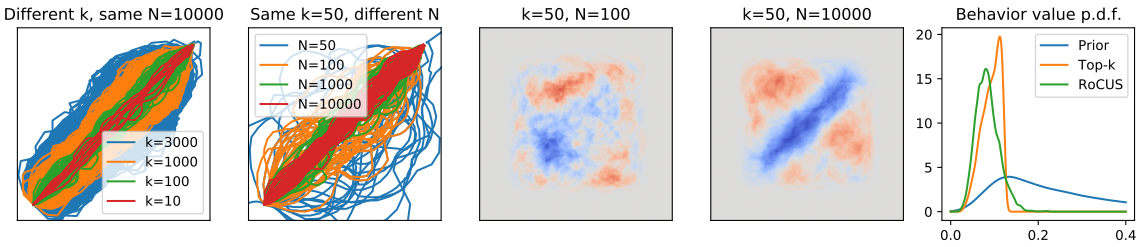


Figure 5-9: Top-k selection baseline. Left two: trajectory distribution; middle two: obstacle distribution; right one: probability density function of behavior values.

but is given equal importance; second, a trajectory even slightly under the threshold is strictly discarded. By comparison, RoCUS gives more importance to more salient samples in a progressive manner, as shown in Figure 5-9 right.

Finally, top-k selection is very computationally inefficient. It discards all of the unselected $N - k$ samples, while RoCUS is much more efficient in that all samples after the burn-in up to the thinning factor can be kept since the posterior concentrated on the salient behavior is directly sampled.

5.7 Discussion and Future Work

RoCUS enables humans to build better mental models of robot controllers. Compared to existing evaluations on task-completion metrics for hand-designed tasks, RoCUS generates tasks and trajectories that highlight any given behavior in a principled way. We used it to uncover non-obvious insights in two domains and help with debugging and improving a controller.

While RoCUS is mainly a tool to analyze robot controllers in simulation as part of comprehensive testing before deployment, it can help understanding (anomalous) real world behaviors as well. When an anomaly is observed, RoCUS can find more samples with the anomaly for developers to identify patterns of systematic failures. Furthermore, RoCUS is not inherently limited to simulation: it only requires trajectory roll-out on specific tasks. For the arm reaching task, this is easy in the real world. For autonomous driving, recreating a traffic condition that involves other vehicles may be hard. However, a key feature of RoCUS is the decoupling of the task

and the controller algorithm, which allows testing on simpler task variants (e.g. with props instead of real cars).

There are multiple directions for future work, including evaluation of *model updates* [16] by defining behavior functions on two controllers, better understanding the samples with explainable artificial intelligence (XAI) methods, and an appropriate interface to facilitate the two-way communication between RoCUS and end-users, as discussed in detail in Appendix C.12.

RoCUS is a framework for systematic discovery and inspection of robotic controller behaviors. We hope that the demonstrated utility of RoCUS sparks further efforts towards the development of other tools for more holistic understanding of robot controllers, and for refining the specifications used to learn or otherwise construct these controllers.

Chapter 6

Model

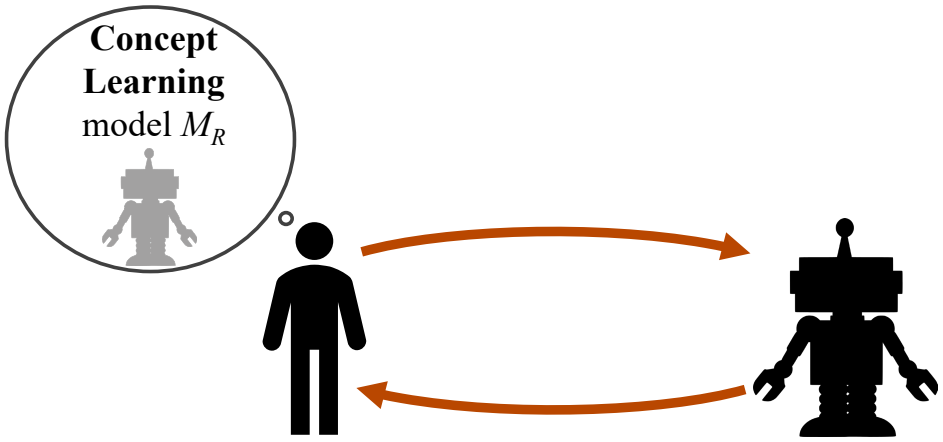


Figure 6-1: This chapter focuses on helping humans build the conceptual models needed to both diagnose specification errors and choose which behaviors to inspect.

This chapter discusses how to assist humans in forming conceptual models to explain the capabilities, limitations, and knowledge of AI systems. Conceptual models interface with both the interactions of specifying behaviors and of inspecting AI system behaviors. In particular, this chapter discusses the processes of "teaching" and "learning" when interacting with an AI system. This language is used to demonstrate the duality: to achieve the fluid interactions described in Figure 1-1 and Figure 6-1, both the human and AI system must learn about each other's reasoning, and both the human and the AI system must equally teach the other to comprehend this reasoning.

This chapter largely reproduces the text from Booth et al.'s HRI 2022 paper *Revisiting Human-Robot Teaching and Learning Through the Lens of Human Concept Learning* [24] and Horter et al.'s HRI 2023 HIRL Workshop paper *Varying How We Teach: Adding Contrast Helps Humans Learn About Robot Motions* [83]

6.1 Introduction

Humans should be able to teach robots new skills, norms, or preferences [88, 3], whether through reward function specifications or other means, but challenges abound. Before teaching, the human can benefit from learning about the robot's current behaviors. Appropriately selecting robot behaviors to show to the human is challenging: observing a robot perform well or poorly biases the human's understanding of its competency [181]. Another challenge arises if the human cannot draw on preexisting mental models for robot behaviors. For example, the human may struggle to learn to predict robot motions if those motions are not human- or animal-like [46]. To assess the impact of their teaching, the human has to compare the robot's current behaviors to its past behaviors—a comparison which is not always straightforward. Bıyık et al. [19] found that humans are unable to provide preferences when robot behavior changes are imperceptible or of roughly equal utility, while Amitai and Amir found that independently selected behaviors are hard to compare [6]. In short, humans find it non-trivial to learn useful conceptual models of robot capabilities and limitations.

We review 35 papers from the human-robot teaching and learning literature, and we contextualize these works by analyzing whether and how they incorporate principles from cognitive theories of human concept learning. Applying these theories supports humans in developing conceptual models of robot capabilities and limitations faster, more accurately, and more flexibly, which in turn can help resolve the aforementioned interaction challenges. Specifically, we look to Analogical Transfer Theory [62, 61], which informs how humans use analogy to transfer prior knowledge to unfamiliar domains, and the Variation Theory of Learning [135, 136], which informs how humans learn to separate superficial details from core knowledge. Together, these complementary theories explain how humans come to understand complex, high-dimensional phenomena and make predictions about unrevealed facts and futures—as such, these theories can be applied to help humans understand robot behaviors. While the HRI community has not previously consulted these theories, each of the works we review inadvertently uses some of their guiding principles.

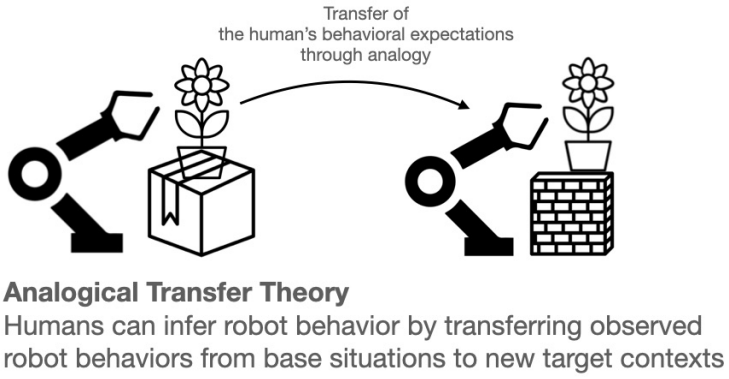


Figure 6-2: Analogical Transfer Theory asserts that analogy is the fundamental process behind conceptual model development. In analogy, people use their knowledge of how a robot acts in one environment to reason about how they might expect the robot to act in a new target environment. Moreover, people use analogy by drawing on past experiences, even with non-robots, to inform their initial beliefs about how the robot might act in any environment. The design challenge when invoking analogical transfer is to guide humans to invoke a correct and useful analogy.

Going forward, humans need interfaces and algorithms that mediate human-robot teaching and learning by *systematically* guiding the human's learning about the robot's behaviors and how they change in response to human input. These interfaces can help humans to (1) learn about the robot's capabilities and limitations, (2) teach the robot by providing a response (e.g., feedback), and (3) learn about the capabilities and limitations of updated robot behavior candidates, and compare these to prior behaviors. Human concept learning theories provide design guidance, i.e., about the selection, sequence, and presentation of robot behaviors, for these interfaces and algorithms. Notably, Variation Theory prescribes an ordered sequence of variance and invariance to help humans distinguish core behaviors from superficial or incidental details, and Analogical Transfer Theory prescribes knowledge transfer by supporting the human's recall with a familiar entity or context.

6.2 Human Concept Learning Theories

Cognitive theories of human concept learning have been refined by testing curriculum interventions. We look to two complementary theories, Analogical Transfer Theory

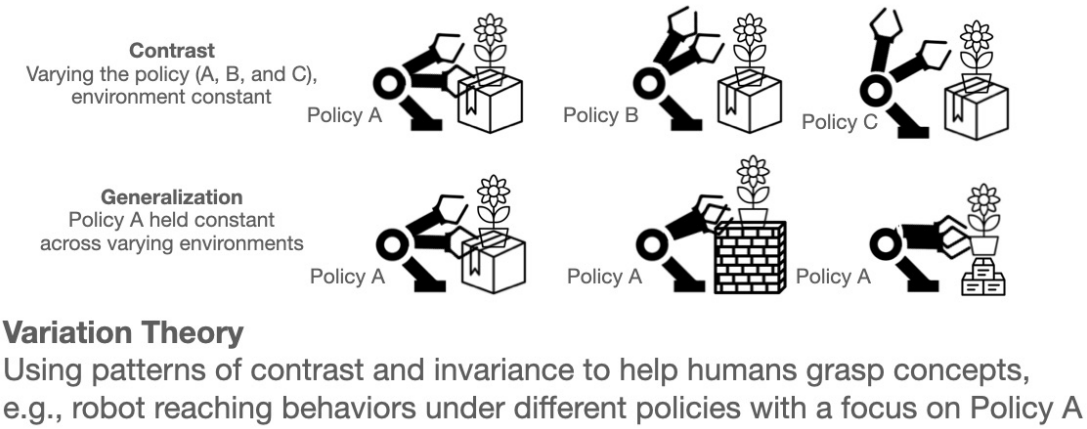


Figure 6-3: The Variation Theory of learning asserts that variation—where people experience structured patterns of differences in both critical and non-critical aspects—is core to conceptual model formation. Variation Theory suggests that people need to experience contrast, where they observe alternative robot policies acting in the environment, to learn about a robot's expected behaviors. The design challenge when invoking variation for concept learning is to identify the critical and non-critical aspects of a learning task and to assess what counterexamples should be used to highlight similarities and differences.

and the Variation Theory of Learning, to inform how interfaces can best mediate the practice of humans learning about robot behaviors. Analogical Transfer Theory explains how humans transfer knowledge to new situations and domains, while the Variation Theory of Learning explains how particular patterns of variation and invariance can help humans discern the difference between superficial details and critical *features* and *aspects*. These processes are key to helping humans understand robots. Figures 6-2 and 6-3 summarize these learning theories graphically.

Analogical Transfer Theory

Studies in cognitive psychology have shown that the parallel presentation of examples helps students attain knowledge gains. For example, simultaneous rather than sequential comparison has been shown to help mathematics students achieve greater gains in both procedural and conceptual knowledge [174]. When analogical encoding, or learning by drawing comparisons across examples, was incorporated in a negotiation strategy curriculum, Gentner et al. [60] found that comparing two parallel cases

rather than studying the cases separately improved schema abstraction and transfer among novices, and that asking learners to describe the commonalities between these cases had the biggest positive impact. These controlled studies are examples of the body of work that collectively informs Analogical Transfer Theory.

Analogical Transfer Theory asserts that *analogy*, or finding and using relational commonalities, is a building block of human concept learning [62, 112, 61]. In analogy, a familiar base domain informs inferences about an unfamiliar target domain. First, a person identifies a candidate base. The person then maps the analogy by *structurally aligning* the base and target; this alignment should highlight relational similarities. Lastly, the person must evaluate the analogy by assessing any inferences drawn from it. Structural alignment and analogy allow people to form new inferences about novel targets (**inference projection**), construct new schemas or mental models by mapping relations (**schema abstraction**), detect differences between bases and targets (**difference detection**), and re-represent bases and targets at alternate levels of abstraction, making the analogy more applicable (**re-representation**).

Analogical Transfer Theory can inform HRI interface design. When faced with a novel domain, people implicitly seek a comparison base domain from memory and search for commonalities between the target and base. When forming an analogy, the person's understanding of the target is bolstered by these commonalities. There are two notable opportunities for interfaces to assist in analogy formation. First, humans are bad at recalling analogous base cases from memory [64], so an interface has an opportunity to prompt the human to recall a relationally-similar base. Second, analogies rely on structural alignment, which highlights the relational commonalities between the target and the base: an interface can present data in an aligned manner such that humans are more readily able to draw inferences about the target or to detect differences.

Variation Theory of Learning

Controlled studies in cognitive psychology have shown that presenting strategically varied examples improves learning outcomes. Students studying high-variability ge-

ometry problems required less mental effort than those studying low-variability examples, and their transfer performance was better and less effortful [160]. When tasked with solving statistical word problems and given either one or three examples with varying or constant superficial details, students with *multiple parallel examples that emphasized structural commonalities by varying superficial details* did best [165]. This variation positively impacted students' schema construction; interestingly, the impact was greatest for those with the least prior mathematical knowledge. Strategic variation illuminates otherwise difficult-to-discern latent structure of concepts [196]. These studies support the potential for variation to help end-users understand feasible robot behaviors.

Variation Theory argues that a person must first discern *critical aspects* and *features* to comprehend some object of learning. Aspects are parameters (e.g., color) while features in this context are instantiations of aspects (e.g., the color red). Aspects are critical when strictly necessary to understand the concept. To achieve robust discernment and learning, the person must experience variation across critical and non-critical (or superficial) aspects. To apply Variation Theory, we designate some aspect(s) as the focused object of conceptual learning. Variation learning then follows an ordered sequence of structured patterns of variance and invariance. These patterns support inductive reasoning to help humans more accurately infer how focused aspects contribute to the object of learning, e.g., a particular robot behavioral policy. For each focused aspect, Variation Theory prescribes the following sequence:

1. **Repetition.** All aspects are held constant. E.g., to learn about a robot's behaviors, the human sees the robot repeatedly act in the same environment.

2. **Contrast.** The focused aspect varies while other aspects are held constant. E.g., Fig. 6-3, the policy varies while the robot operates in a fixed environment.

3. **Generalization.** The focused aspect is held constant, while other aspects vary. E.g., Fig. 6-3, the human sees how the robot's policy varies in new environments using the selected value of the focused aspect.

4. **Fusion.** All aspects vary to mimic "real world" variation.

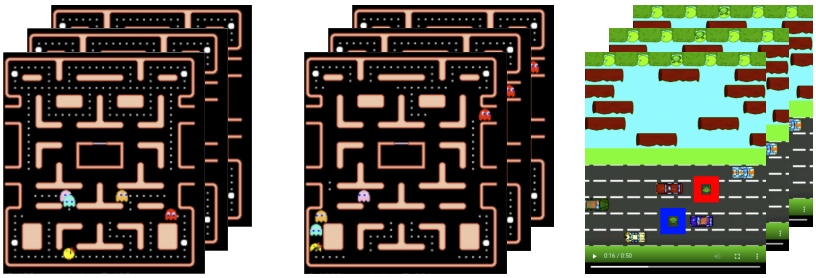
Variation Theory has been used effectively in many domains, including story comprehension [196], learning vocabulary words [48], Chinese characters [116], the color of light [126], mathematics education [138], chemistry education [31], and computing education [188]. Books have discussed how Variation Theory can improve teaching and learning in schools [127, 137]. In HRI, Variation Theory has immediate application: many interfaces solicit human feedback as reflections on single executions of robot behaviors. But this fails to accommodate the backbone of Variation Theory: to provide high quality feedback, the person needs to understand the robot's behavior—which means they need to understand the underlying critical aspects of the robot's behaviors by first experiencing variation of the underlying critical features—*before* providing preferences or feedback over these behaviors.

Concept Learning for Hypothetical Robot Applications

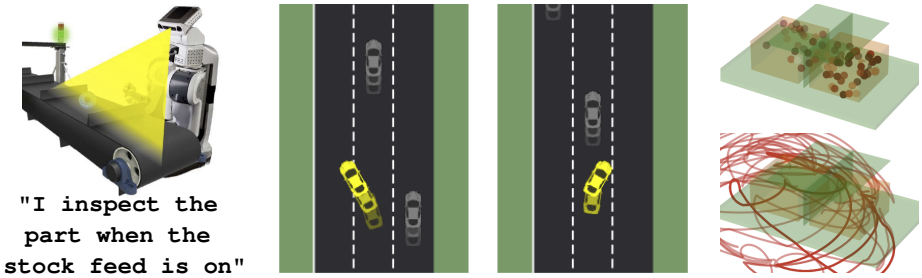
In HRI, human concept learning occurs whenever the human must learn about robot behaviors. We consider two hypothetical robotics applications for exposition.

Consider collaborative assembly. Traditionally, robots halt whenever a human enters a shared work region. Modern approaches let robots predict human behavior and optimize their plans to increase system uptime [198]. For successful collaboration, the human should also learn about the robot's behaviors, both to increase their comfort in proximity and to help optimize robot uptime. For this, the human benefits from understanding both the robot's workspace and motion planner. A naive approach might show a human the limits of the robot's workspace or an example motion. In practice, however, the effective working patterns of the robot seldom reach these limits, and the human's learned conceptual model would be too conservative. By instead applying Variation Theory and experiencing variation in the robot's positioning and motions, the human can learn a conceptual model of the robot's effective workspace. With this, the human can feel safer (by predicting how the robot will move), and work to increase the robot's uptime (by avoiding interfering with the robot's plan).

In other robotics applications, the robot may have frequent, fleeting interactions with non-expert users—for example, a delivery robot needs to navigate alongside



(a) Side-by-side [4, 181] (b) Axis aligned [6]



(c) Predicates [76] (d) Critical states [84, 158] (e) Overlaid [213]

Figure 6-4: Policy summarization. (6-4(a)) uses contrast through side-by-side video summaries of varied policies [4, 181]. (6-4(b)) better aligns differences with videos of two varied policies—the red and blue agents—in an axis-aligned, shared state [6]. (6-4(c)) presents logical statements with states grouped by Boolean predicates [76]. (6-4(d)) presents varied critical states [84, 158]. (6-4(e)) overlays visuals to structurally-align both varied environment configurations (above) and trajectories (below) [213].

pedestrians and others. In such applications, particularly when humans have only brief encounters with robots, Analogical Transfer Theory can assist. One challenge with delivery robots is their potential use of omni-directional wheels: while these wheels provide flexibility to a large range of motions, humans experience discomfort when interacting with such a robot, since the motions these wheel structures exhibit are hard to predict [102]. One way to apply Analogical Transfer Theory is for the robot designer to leverage physical analogies: if the robot closely resembles a car (as is common in such applications), humans who interact with the system would anticipate car-like motions, which are dissimilar from many omni-directional wheel motions. Styling such robots after humans encompasses a larger range of acceptable omnidirectional motions, but challenges remain in aligning these motions to human expectations [102].

6.3 Concept Learning in HRI

We review 35 papers from the literature on human-robot teaching and learning. We study how these works benefit from human concept learning principles, and how better curriculum design could support more effective interaction. Though these roles are inherently fluid, we focus on works in which the human is primarily the "teacher" and the robot the "learner"; i.e., we exclude robot tutoring. We selected works which have the following goals: policy summarization, updating human beliefs, or teaching with feedback, preferences, and/or corrections. The first two goals implicitly build curricula for informing conceptual models of robot behaviors; the latter three help humans teach robots with seemingly intuitive signals. We selected papers primarily from premier venues (e.g., HRI, NeurIPS, AAAI), which are highly topical or influential in these niches (e.g., #citations$\geq$50). See the supplementary material for further analysis and the appendix for a definition of "objects of learning" in these settings.

6.3.1 Policy Summarization

Policy summarizations aim to help a human understand the robot's expected behaviors [5], allowing the human to determine an apt level of autonomy to afford the robot. Fig. 6-4 shows example policy summary interfaces.

Implementations

Several policy summarization methods [84, 4, 6, 181] use $\mathcal{Q}$-values to select informative states; these quantify the benefit of taking action a in state s. One approach selects states with the largest delta in $\mathcal{Q}$-values across actions [84, 4]. Huang et al. [84] call these *critical states* as they support learning about the robot's capabilities and limitations. In concept learning, critical states are *critical features*: a person cannot learn about the policy without understanding the robot's behaviors in these states.

Another method requires shared Boolean predicates between human and robot [76]. This method takes predicate-annotated trajectories, and solves a set cover problem over these traces to answer questions like, "When does the robot do a?" with answers

like, "The robot does a when p or q." Another approach presents counterfactuals by finding similar states which elicit different actions [158], and a final method applies Bayesian inference to find environments where the policy expresses a specific property, like maximal directness [213].

Analogical Transfer Theory

Policy summarization approaches make extensive use of structural alignment, the backbone of Analogical Transfer Theory. These works all facilitate schema abstraction, wherein the human assesses how the policy would apply in new target environments. Several approaches use structural alignment by visualizing shared states with different policies side-by-side [4, 181, 84] or overlaid [6]. Hayes and Shah [76] use shared predicates, and present textual summaries of these groupings to a user for both schema abstraction and inference projection, wherein a user learns how the agent will behave in a new state with shared properties. Countering this, Zhou et al. [213] instead structurally align environments and trajectories (but not individual states) by overlaying semitransparent visualizations to support schema abstraction for policy failure modes.

Several approaches also support difference detection, though to differing levels of success. Some compare multiple candidate policies [6, 4, 181, 84]; the goal is to assist a user in selecting the best policy. Most such methods find interesting states for two or three policies independently, and present these states or behavior samples for each policy side-by-side [4, 181, 84]. However, this independent search does not maximally find differences between these candidate policies. As such, these policy candidates are not well structurally aligned, and these side-by-side comparisons may be difficult or impossible for the user to assess. Amitai and Amir [6] propose a counter approach: instead of generating interesting states for each policy independently, they simulate both policies in parallel, and find policy disagreements, where the respective policies choose different actions in a shared state. They then show these two different policies in an axis-aligned manner, such that the user is more readily able to detect differences and select the more appropriate policy for their intended context.

Variation Theory

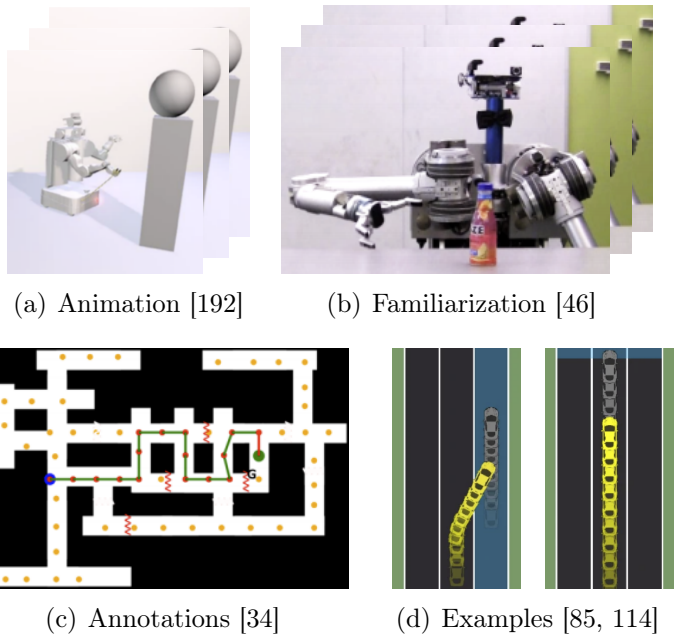
Using Variation Theory's contrast, Hayes and Shah [76] group states (through Boolean predicates) while maintaining a fixed policy and a fixed action. Zhou et al. [213] simultaneously present varied environments and trajectories, again for a fixed policy. Huang et al. [84] and Olson et al. [158] similarly present varied states as focused aspects, while maintaining a fixed policy. For fusion, Huang et al. [84] additionally compare policies; in this, every aspect is varied. Similarly, several other approaches [4, 6, 181] present a varied set of states alongside two [4, 6] or three [181] candidate policies. While none of these works use the language of Variation Theory, these incorporated patterns of variance and invariance help these works achieve their goals of supporting the human in learning about the robot's policy.

Takeaways and Frontiers

Sequeira and Gervasio [181] found that users often anchor their perceptions of a robot's capabilities on their initial experiences [57]; as such, they found the importance of "appropriate" variation, and they employed Variation Theory's fusion as a learning strategy. Nonetheless, they do not define an "appropriate" amount of variation. Variation Theory can help: it can guide the design of strategic and sufficient variation to support the human's innate learning abilities. A more structured exposure with Variation Theory's prescribed sequence of contrast→generalization→fusion could improve the human's learning. *Fusion isn't all we need.* Learning to discern the limits of robot behaviors, through contrast and generalization, is needed, too.

6.3.2 Prompting Human Belief Updates

Many methods explore how to prompt humans to update their beliefs. These include generating expressive motions [192, 113, 46] or state/action pairs [84, 114], or constructing "patches" to reconcile divergent models [34]. See Fig. 6-5.



(a) Animation [192] (b) Familiarization [46]

(c) Annotations [34] (d) Examples [85, 114]

Figure 6-5: Interfaces for updating human beliefs methods. 6-5(a): [192] uses re-representation by using animation principles to induce legibility through anthropomorphized familiarity. 6-5(b): [46] incorporates generalization by showing a curriculum of varied motions. 6-5(c): [35] uses difference detection, wherein robots provide annotations to correct a human's model—here, as a map. 6-5(d): [85] and [114] use generalization: they show varied trajectories while holding the policy constant.

Implementations

Takayama et al. [192] observed that robots require time for planning, but the transition between "thinking" and acting often catches humans off guard. To address this, they used animation principles of anticipation and reaction for more expressive motion. Kwon et al. [113] observed that robot failures are not expressive—typically, the robot just stops. They generate expressive trajectories which mimic successful trajectories *in spite of failure*. Lastly, Dragan and Srinivasa [46] studied whether humans could learn a robot's motions through familiarization. They optimized motions with two cost functions: one which enables human-like motion; another which enables unnatural motion by prioritizing shoulder motion over wrist motion. They discovered users were more adept at predicting natural motions in new settings.

Huang et al. [85] and Lage et al. [114] seek expressive states which allow a person to update their beliefs about the robot's objective. Huang et al. [85] assumed the

human uses inverse reinforcement learning (IRL) to model the robot's objective, and then used Bayesian inference to find environments which are maximally informative to the human's beliefs. Lage et al. [114] compared IRL with imitation learning. Both assess the human's learning by testing their knowledge in unfamiliar contexts. Finally, Chakraborti et al. [34] considered a search-and-rescue task, where the human has an outdated mental model of the environment while the robot acquires knowledge of how a disaster changed the environment. To update the human's beliefs, the robot explains model differences by providing a patch expressing why its new plan is acceptable.

Analogical Transfer Theory

Takayama et al.'s work [192] is unusual and interesting in its use of re-representation from Analogical Transfer Theory. In re-representation, a base and/or target is re-represented at a higher level of abstraction so a user is more readily able to perform analogical reasoning. They use animation to re-represent the robot as a more familiar entity to communicate robots switching from planning to acting. In using anticipation and reaction animations, they structurally align the robot's motions to those of anthropomorphic characters. This draws on humans' intuitive understanding and helps the human extrapolate their understanding to the robot by analogy.

The other human belief update works only lightly use Analogical Transfer Theory. Kwon et al. [113] used difference detection by structurally aligning—as much as possible—a failed trajectory to an imagined successful trajectory. From this, the human learns about the delta between these trajectories, which helps them comprehend the robot's failure. Chakraborti et al. [34] also drew on difference detection by assuming that the human and robot have divergent but partially-aligned models of the environment and aligning differences with model patches, which aim to reconcile the human's model to match the robot's. Lastly, several methods [46, 85, 114] employed both inference projection and schema abstraction, though with minimal structural alignment: they presented similar training environments before assessing the humans' abilities to project in a slightly unfamiliar environment.

Variation Theory

For generalization, Dragan and Srinivasa [46] discretized a robot's goal space and generate motions for each goal. The human learned through familiarization: they showed the human examples of the robot's motions with varied goals while fixing the underlying policy. They report that familiarization—using generalization—improves the human's accuracy in predicting the robot's motion, but not as substantially as anticipated. To test humans' accuracy, this study asks users to select a motion trajectory from three different choices, each generated by a different policy. This choice relates to the principle of contrast, though it is used only to *test* the human and not to *teach* the human about the robot's behaviors. Variation Theory shows contrast should precede generalization: to learn about the robot's policy, the human might benefit from seeing the results of *varied* policies as focused aspects before seeing the results of varied environments.

Huang et al. [85] and Lage et al. [114] incorporated generalization when teaching the human about a robot's objective. Both methods hold the policy, the focused aspect, constant while varying the environment. Lage et al. [114] additionally incorporated fusion by showing multiple varied policies side-by-side. Countering this, Chakraborti et al. [34] primarily used contrast. In their approach, the object of learning is not the policy, as is typical; instead, their object of learning is the true environment. They assume the human and the robot have different models of the environment—effectively, varying the environment. They reconcile these differences by updating the human's model with patches which explain the robot's model.

A Follow-Up Familiarization Study: Adding Contrast

In a follow-up study [83], we reimplemented a similar protocol to Dragan and Srinivasa [46]. Specifically, we assessed the effects of adding contrast, where users see both correct and incorrect robot motions, to the training procedure for learning about robot motions. We implemented this study on a Franka Panda robot arm.

In both the generalization and contrast study arms, participants watched 14 videos

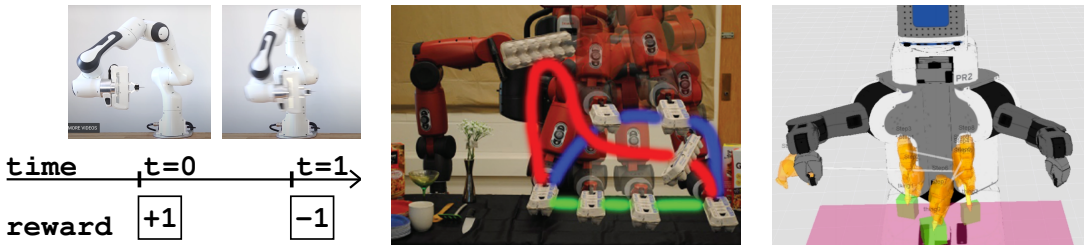
of robot motions. With generalization, each video corresponded to the robot moving from a fixed start location to a different target location (i.e., 14 target locations). With contrast, the human saw side-by-side videos of both unnatural motion controllers with one indicated as correct. Since each contrast target consists of two videos, we showed only 7 target locations in the contrast condition.

In our reimplementation of Dragan and Srinivasa's generalization (or, familiarization) protocol, users were 52.4% accurate in predicting robot motions. With added contrast, users' accuracy rate increased to 70.2%. A two-sample T-Test indicates we cannot reject the null hypothesis that the mean accuracies between generalization and contrast are the same: $t(38)=-1.43$, $p=0.08$. While not statistically significant, the large accuracy improvement suggests the contrast intervention has promise. Moreover, adding contrast did result in a statistically significant improvement in novel settings, wherein the user had not seen the particular target during training. For novel settings, the contrast users exhibited higher accuracy (72.4%) than generalization users (50.0%). A two-sample T-Test indicates that we can reject the null hypothesis: $t(78)=-2.35$, $p=0.01$. This accuracy improvement may be a consequence of participants using contrast to establish concept boundaries, improving their ability to infer constraints of motion in unseen dimensions of variation.

Variation Theory asserts that contrast should be experienced before generalization, but that the combination of these steps should result in the greatest learning gains. Studying how these interventions can best be combined in many domains is a compelling direction for future work.

Takeaways and Frontiers

Analogical reasoning is an especially useful tool for assisting in the task of guiding humans to update their beliefs about a robot, especially in new encounters. The goal in such settings is to push the human to establish correct assumptions; analogy can rapidly accomplish this. Takayama et al.'s [192] approach of using animation principles to change how the human understands the robot is compelling, as this uses re-representation through anthropomorphic characteristics. Other works have



(a) Feedback (e.g., [107]) (b) Preferences [90] (c) Corrections [2]

Figure 6-6: Teaching with reward (6-6(a)), preferences (6-6(b)), and corrections (6-6(c)). With reward (6-6(a)), the person observes the robot and rewards it for past actions, e.g., with a ± 1 signal. With preferences (6-6(b)), the person selects between two or more candidate trajectories; the example in 6-6(b) structurally aligns three trajectories, supporting difference detection. Finally, with corrections (6-6(c)), a person modifies past trajectories—either through a GUI [2, 55] or physical manipulation [15, 14].

explored the related question of how robots can benefit (or suffer) from playing to stereotypes [193]. This presents a frontier for efforts to support humans in learning about robots: designing and structurally aligning robots to appropriate analogical bases—whether through animation, stereotype play, visual design, or other means—can support the humans' belief update process and human-robot teaching and learning interactions in general.

6.3.3 Teaching with Feedback

Fig. 6-6 shows interfaces for intuitive teaching. When teaching with feedback, a human gives feedback as they watch an agent act. These works are motivated by the idea that "you know it when you see it." This assumption seemingly undermines the need for human concept learning: if a human *knows it* just by seeing, why should the human first need to learn about the robot's capabilities and limitations? For some tasks, a human is able to provide feedback with little learning. For example, if giving feedback to a simulated car, a human might say "Good robot!" when on the road, or "Bad robot!" otherwise [171]. For other tasks, though, a human might lack intuition for the constraints of a robotic platform, and give poor feedback due to malformed expectations, e.g., they might believe the robot can learn a behavior

which its morphology cannot accommodate [35, 113]. Or, the robot might be making progress toward a satisfactory-but-unexpected policy (as seen in, e.g., [33]). In these cases, the human must learn about the robot's capabilities and limitations *before* providing feedback.

Implementations

In TAMER, a human-supplied reward signal is used to train a supervised learning module, which approximates the human's reward [107]. TAMER was first tested in simulated Tetris and Mountain Car environments [107], and later evaluated in robotics settings [109]. A modification to TAMER biases the agent toward taking non-optimal actions for the sake of stimulating human engagement [104].

The Advise method instead interprets feedback as a commentary on actions: reward is determined by the environment, and feedback is used to guide action exploration. Griffith et al. [70] demonstrated Advise on simulated game environments with simulated teachers, while Cederborg et al. [33] demonstrated Advise in a user study. Curiously, the real users outperformed the simulated users as real users were able to adapt to different but equally good strategies, while simulated users provided negative feedback if the behavior did not match their pre-planned strategy. The real human might also have preferred a different policy; nonetheless, they learn about the agent's policy and adapt to its learning trajectory.

TAMER and Advise do not consider the *teacher's strategy*. Loftin et al. [129] conducted a user study showing that some users bias toward giving positive feedback while others are more balanced. They introduced SABL: a method which numerically manipulates human feedback based on the teacher's strategy; this strategy is learned during interaction. A similar approach, COACH, uses the insight that human feedback depends on the robot's current aptitude [133]. Accounting for this policy-dependency in feedback, COACH interprets feedback as an advantage function. COACH assumes that the human teacher is learning over time: to give feedback, the human must first learn about the current policy.

Analogical Transfer Theory

None of these works extensively use structural alignment—instead, these all require prolonged observation to learn about a policy. Nonetheless, COACH uses difference detection [133], though without the structural alignment recommended by Analogical Transfer Theory. COACH assumes that human feedback is policy-dependent, varying as the policy improves or degrades over time, and difference detection is needed to assess how the policy changes. This presents an opportunity for the application of Analogical Transfer Theory. Instead of tasking the human with watching the robot repeatedly attempt a task, and hope that the human identifies the differences or similarities over different trials, applying structural alignment would help. After policy updates, a supporting communication and intervention interface could provide highlighting or other means of identifying differences to support the human's assessment.

Variation Theory

In feedback user studies [107, 42, 125, 33, 129, 133], variation is implicitly present, though not espoused as a core principle to support human learning. These works assume that a human is able to watch the agent act and give appropriate feedback in response. As the human observes, they are presented with varied environments, states, and actions in all cases. For several of these approaches [107, 42, 125, 129, 133], the policy is updated in real-time in response to the human's feedback, and is therefore also varied simultaneously. All of these examples use fusion. Countering this, Cederborg et al. [33] hold the policy constant during feedback collection, and thus supports generalization, instead. They do so to support different experiment conditions, but this may coincidentally increase the human's aptitude for teaching, as they are better able to learn about the effects of the policy. To better support human concept learning, future assessments and algorithms should present this variation with a deliberate and managed approach. This is an opportunity for future research.

Takeaways and Frontiers

While none of these feedback-based approaches extensively incorporate principles from either theory of human concept learning, "you know it when you see it" isn't enough to support human-robot teaching and learning in the forseeable future. Both SABL [129] and COACH [133] observe that learning from feedback cannot be formulated as a person- or policy-independent algorithm; nonetheless, the tasks tested in all of these feedback works conform to the expectation that the human can give good feedback after short periods of unstructured observation. As these techniques accommodate increasingly complex tasks, and as the features used to complete a task further diverge between humans and robots, this assumption will ring hollow. Using human concept learning theories—particularly by using variation to communicate the behaviors of the current and future policies—can mitigate these challenges.

6.3.4 Teaching with Preferences

Implicitly, preferences mandate that interfaces present multiple options: does the human prefer A or B? In this manner, preferences naturally rely on concepts of variation, and the requisite structural alignment supports the human in teaching.

Implementations

Sadigh et al. [178] take an active learning approach to selecting trajectory pairs for soliciting human preferences. They formulate this as volume removal over the distribution of potential reward functions, wherein each preference should maximize the volume removed from this hypothesis space. Their interface asks humans to compare trajectories generated by different policies in the same scenario. Bıyık et al. [19] observe that volume removal can fail to support human teaching as the robot can ask the human to compare two trajectories with imperceptible differences. They instead introduce an information gain approach for trajectory selection; this approach selects queries by maximizing both the robot's uncertainty over the human's response and the *human's uncertainty* in providing a preference.

Instead of generating trajectories from different policies, Christiano et al. [39] roll out the same policy numerous times in slightly varied environments. Stochasticity in the policy, transition dynamics, and environment introduce variation. They similarly use active learning to select trajectory clips which are maximally uncertain under their reward model. Ibarz et al. [87] uses this same method, but, instead of learning from tabula rasa, they initialize their agent through imitation learning and then use preferences for policy refinement. Curiously, they found active querying did not increase the agent's learning performance, while slowing down sampling. They thus opted to adopt random sampling for trajectory clips instead.

Jain et al. [90] formulate learning from preferences as an iterative process. They observe that humans are typically unable to provide optimal demonstrations, but can re-rank trajectories iteratively. For this, they learn a model of a user's scores for trajectories. To generate trajectories for comparison, they fix the starting and ending states, and use a rapidly-exploring random tree planner with heuristics to encourage diversity. Lastly, Wilson et al. [201] approximate a policy distribution using a Bayesian likelihood function. Using this distribution, they sample two policies and generate two trajectories for comparison. They compare two active approaches for selecting policies for comparison. First, they consider policies which generate different behaviors when rolled out. Second, they consider the expected belief change in the hypothesis space.

Analogical Transfer Theory

Preferences naturally incorporate structural alignment. If two choices are not structurally aligned, it can be challenging or impossible to discern differences—a prerequisite for providing preferences. While most of these works engage structural alignment and difference detection by presenting trajectories side-by-side [178, 39, 87, 19], they do not take full advantage of Analogical Transfer Theory. Christiano et al. [39] and Ibarz et al. [87] present side-by-side trajectory snippets with differing start and end states; these differences make the snippets are hard to compare. Most notably, Jain et al. [90] present figures which show structurally-aligned, overlaid trajectories with

shared start and end states (Fig. 6-6(b)), but, in their experiments, humans watched a robot perform trajectories sequentially, and were then asked to rank them. These authors cite this as a limitation of their work: they note that making users memorize these trajectories and not aligning them hinders the efficacy of their approach.

Variation Theory

In all of these works, variation in trajectories is a prerequisite for *robot* learning. Through experiencing this variation, the human is also better equipped to understand the robot's policy. These works all incorporate contrast [178, 19, 90, 201] or generalization [39, 87]. They support contrast by varying the underlying policy [178, 19, 90, 201]—whether by varying their parameterizations (e.g., [178]) or by using an alternative for comparison (e.g., [90]). To support comparisons between policies, these works hold all other aspects invariant—e.g., the starting state, and sometimes the end state [90]. These works incorporate generalization by requesting preferences over multiple trajectory segments from the same policy [39, 87]. The same policy may also present different trajectories for comparison, as the policies and transition dynamics may be stochastic.

Takeaways and Frontiers

Teaching a robot with preferences naturally incorporates principles from both Variation and Analogical Transfer Theory. Variation is implicitly present, as the person is tasked with choosing between two or more options. Analogical Transfer Theory is also incorporated through the use of structural alignment, either presenting choices side-by-side or overlaid to support difference detection. In these settings, structural alignment is often still hard to comprehend, and could be improved through by *maximizing* this alignment—starting by presenting trajectories with shared start and/or end states. Despite this natural proclivity for engaging human concept learning, preferences approaches again commonly assume that the human either learns about the robot's policy from observation or is able to give feedback without any context. This is a missed opportunity.

6.3.5 Teaching with Corrections

Lastly, we consider teaching with corrections. A robot starts with an initial policy, and the human is tasked with correcting it—e.g., by teaching the robot about preferred action choices.

Implementations

Alexandrova et al. [2] introduced a corrections interface where a user first provides a demonstration to a robot and subsequently corrects its policy. This interface is gnarly and complex: after providing demonstrations, users can modify past demonstrations by changing frames of reference or by deleting intermediary poses and/or landmarks—all from the robot's point of view. They found that visualizing the robot's learning was extremely useful, and deleting poses was also helpful. Using the same interface, Forbes et al. [55] sourced corrections from a crowd.

Bajcsy et al. [15] reframed corrections from the perspective of *physical* HRI. Humans often physically engage with robots—for example, by pushing it out of the way. These interactions are typically regarded as disturbances, but Bajcsy et al. noted some useful information. They introduced an optimization approach to update the robot's trajectory to align with the human's physically-corrected trajectory. In subsequent work, Bajcsy et al. [14] introduced a method where instead of using the full, corrected trajectory as the optimization target, they allow only one feature to vary at a time.

Analogical Transfer Theory

Correction-based systems naturally incorporate difference detection. In all of these approaches, the robot starts with some trajectory which needs to be corrected. To support difference detection, this trajectory is structurally-aligned with a corrected trajectory—either through a visualization to teach the human about what to teach [15, 14] or through an omnipresent interface used for supporting the user to correct the robot's behaviors and evaluate progress [2, 55]. In the former, the difference is only

shown at the beginning of the interaction. While this assists the human in learning about the task expectations, it requires them to recall the behavior. A better interface supports and maintains this visualization throughout the interaction.

Variation Theory

Alexandrova et al. [2] found that repetition is remarkably beneficial for corrections-based teaching: in their system, a human trains a robot through demonstration and subsequent corrections in a GUI. They find the mere presence of these visualizations and the ability to repeatedly observe actions to be the greatest benefit to humans' teaching. The repetition step of Variation Theory is often overlooked or skipped—but this result suggests it can be an effective supporting methodology. Bajcsy et al. [15] incorporated fusion in their first approach: they tasked a human with physically manipulating a robot to correct its expressed trajectories, where the robot learns from these corrections. Bajcsy et al. [14] instead use contrast, wherein the robot isolates updates to the single feature which changed most in the human's corrections. They compare their fusion and contrast implementations, and find the latter to be more effective for the robot's learning.

Takeaways and Frontiers

Corrections are a powerful teaching tool. Intuitively, we might expect humans would prefer to correct every aspect of a robot's behavior simultaneously; after all, that is efficient! In practice, Bajcsy et al. [14] demonstrate that this assumption is flawed—and their implementation reflects Variation Theory's insights. They find that humans are in fact more adept at correcting a trajectory by varying *one feature at a time*. Although this work is framed from the perspective of robot learning, Variation Theory suggests its implications will also hold for the inverse, the human's learning. By iteratively changing one feature at a time, both the human and robot learn from a varied critical aspect, while holding all else invariant. This approach supports the human in discerning the impact of that individual change.

6.4 Design Guidance & Future Directions

Human concept learning provides a new lens for human-robot teaching and learning. Without explicitly consulting these theories, past approaches have incorporated a number of their insights. Still, many gaps and opportunities remain.

Supporting Analogical Transfer

When teaching a robot, humans are likely to employ analogy to inform their beliefs about the robot, as well as their beliefs over how the robot will use their teaching signal to change its behaviors [16]. Humans might use any number of bases to inform their interactions: human or animal behaviors, virtual character behaviors, or past experiences with other robots. Only three systems we analyzed considered base case retrieval as a design input [192, 46, 113] by using exaggerated, anthropomorphic, and/or animated behaviors. Future efforts in human-robot teaching and learning should build on these ideas, and provide further support for base anchoring: instead of giving the person independence in selecting their own base, the presentation of the robot should guide the person to select an appropriate and desirable base.

Analogy's backbone is structural alignment. This is used throughout many of the human-robot teaching and learning systems we analyzed, usually to support difference detection. These prior works often assess whether a human is able to perceive some difference or provide some teaching signal [19]; nonetheless, these works rarely considered how to maximally-align information such that the human is best positioned to make these assessments. For example, in asking users to compare trajectory snippets, some works showed trajectories to users that both started and ended in different states, while also expressing variation in the interim [39, 87]. Such tasks ignore structural alignment, and are unduly challenging. Designing for maximal structural alignment is a promising path forward.

Supporting Structured Variation

Variation Theory informs how humans learn to discern the latent structure of new concepts, and to understand the bounds of their applicability. This theory does not, however, inform us of exactly how it should be applied in HRI settings. Specifically, Variation Theory requires the designation of an object of learning and a number of aspects related to that object of learning. These can be inferred, to some extent, from the task structure (e.g., see the appendix). Even so, identifying exactly how to group and present aspects to facilitate human concept learning is a design task, and requires substantial experimentation and prototyping.

Variation Theory then proposes a strict sequence for efficient concept learning: repetition, then contrast, then generalization, then fusion. Despite this, none of the 35 works we looked at followed this prescribed sequence. In policy summarization, Sequeira and Gervasio [181] noted that finding an appropriate amount of variation when using fusion was challenging: too much and users were confused about an agent's capabilities and limitations; too little and users believed agents to be either more competent or less competent than they really are. Using the prescribed structured presentation of variation is uncharted territory in human-robot teaching and learning systems, but it offers a potential resolution to this challenge and may additionally elevate human ability to learn about robots.

In this review, the focus is implicitly on helping the robot learn from human teaching, and not on helping the human be a better teacher. The human is treated as an oracle—able to provide a perfect assessment of behavior at any time. Nonetheless, when variation is used as a tool to guide the robot's learning, the human inadvertently learns too (e.g., [14]). Future algorithms and interfaces should consider this more directly: structured variation can support both the human and the robot in discerning critical aspects, even if these aspects are not the same for both entities. A symbiotic approach to human-robot teaching and learning could optimize the data requirements to satisfy the variation needs of *both* human and robot.

Explainability

In AI and HRI, explanations aim to support debugging, to calibrate end user trust, and to moderate model reliance; these goals make explanations promising for human-robot teaching and learning. Despite the introduction of many methods (of sometimes dubious quality [1, 214]), explanations often do not help people achieve these goals [189, 30]. Engaging human concept learning can help humans use generated explanations effectively. Onboarding for these methods is important [32] but often overlooked [82]. Onboarding can use Variation Theory to help users understand the bounds and limitations of explanations, and Analogical Transfer to help users bootstrap prior knowledge onto these new methods.

Chapter 7

Discussion & Future Work

In this last chapter, this dissertation discusses opportunities for future work to contribute to the joint tasks of assisting people in writing specifications for AI systems and in interpreting these systems' learned behaviors. These ideas are seedlings that, with sufficient care and thought, could flourish into future doctoral theses.

7.1 Revising Specifications

The most obvious directions for future work is to combine the three interaction components discussed at length in this dissertation—on specifying, inspecting, and modeling AI system behaviors—and evaluate whether the tools and insights presented are confirmed to be beneficial for ultimately revising specifications. Setting up an appropriate evaluation of this larger interactive system is itself a non-trivial research task, in part because there are not yet standardized tests or metrics to assess alignment—though some have been proposed [29, 179]. One simplifying proposal is to omit the requirement that people revise their specifications, and instead provide them with specifications with known errors. Like the work presented in Chapter 3, this simplification removes ambiguity about whether the drafted specification is correct or aligned. In this simplified proposal, the research question is whether people can use the tools presented in this dissertation to identify the known flaws in the given specification.

To validate this claim of improved alignment more generally, this assessment

should be studied across many domains, and some of these domains will unearth research challenges that need to be addressed through substantive modification. Some challenges are predictable; for example, the work on inspecting behaviors—in Chapters 4 and 5—make assumptions that will be difficult to remove. Both Chapters 4 and 5 assume the existence of generative models or simulators with sufficiently compact latent spaces such that these latent spaces can be efficiently sampled using MCMC methods. While generative modeling techniques are becoming ever-increasingly rich, these models are nonetheless still imperfect and will inevitably lead to failures in some domains. Chapter 5 is additionally stressed by concerns of sim-to-real translation, a notoriously hard problem in robotics [163]. Solving these challenges may require the design of alternative protocols for inspecting behaviors for some domains.

7.2 Building Conceptual Models

This dissertation conducts a meta-study to assess how Human-AI interaction works integrate principles from human concept learning theories, as presented in Chapter 6. This is a rich study because it considers a vast landscape of research. However, it is also limited because seemingly none of these studies directly sought to incorporate human concept learning. This dissertation also contributes a preliminary study, in a single domain, of the benefit of adding a single component of the Variation Theory of Learning: contrast. This leaves low-hanging fruit for future research: we should study how the addition of each component of each concept learning theory affects a person's ability to write specifications—both independently, and in combination. By conducting this study directly, we can isolate the contribution of each component. However, this research endeavor is necessarily challenging: while these theories of human concept learning and this dissertation's analysis of them provides guidelines for interaction design, implementing these recommendations nonetheless requires significant creativity, and the design space of possible implementations is vast.

7.3 Inspecting "Interesting" Behaviors

The work on inspecting behaviors in Chapters 4 and 5 asks the human to craft a query in the form of a metric for behaviors. For example, for neural network settings, these metrics consist of a prediction probability as well as constraints over the latent space of the accompanying generative model. For robot controller settings, these metrics consist of numerical scores of behaviors—like how indirect the controller is in its motion. These chapters demonstrate many such metrics and show how these selected metrics can be useful for conceptual model formation and system debugging. Nonetheless, choosing these metrics is itself a design challenge and research question. What behavior is expected to be interesting, and why? Future work could seek to automate these queries, or to guide the human in designing queries that are likely to expose interesting behaviors. Chapter 6 provides some insights on how to approach this problem—for example, by using coupled, contrastive metrics and grouping samples to better expose system behaviors. The broader meaning of "interesting" in the context of AI system behaviors is again a research question.

7.4 Learned Reward Functions

This dissertation, and particularly Chapter 3, is largely predicated on the idea that a human is hand-crafting a reward function as a specification. This setting has many nice properties: because an expert is designing the reward function, there is implicitly an expectation that the reward function will be scrutable and that the expert can be held accountable for the actions of their designed system, at least to some extent. While hand-designed reward functions are widely used (e.g., [205]), with the rapid success and growth of ChatGPT, which uses a reward function learned from human preferences to assess the relative goodness of potential text outputs, there is an increasing push for their replacement with learned reward functions [159].

Learned reward functions are sometimes seen as a catch-all shortcut solution to the challenges of writing high-quality specifications—but, of course, this is not the case.

First, learned reward functions compromise on scrutability, which is a significant loss. Second, my coauthors Knox et al. and I [106, 105] showed that the question of whether the underlying correct reward functions can be recovered by common reward function learning paradigms depends on the input assumptions about how human preferences are generated. This dissertation encourages further study on this topic: how do the findings of Chapter 3 carry over to learned reward functions? In particular, are learned reward functions generally correct, or do they too propagate systematic errors? And, are learned reward functions equally subject to the problem of overfitting? These questions are compelling directions for future study!

7.4.1 Adding Intuitive Teaching Signals

One significant adaptation to the iterative vision described in Figure 1-1 concerns the form of specification revisions over iterations. Perhaps the first specification is a hand-designed reward function, or a reward function learned from a dataset of preferences. On a future iteration, it might be sensible for the human to provide an intuitive teaching signal—like corrections, feedback, or preferences—to update the specification. Should this modification affect how we reason about alignment, or about the processes of specifying, inspecting, and modeling the AI system's behaviors? Equally, we have thus far discussed a single human as part of the iterative system, but this human could be replaced—for example, by someone with less expertise, like an end user of the system. The human could instead be replaced with a plurality of end users, as in a deployed system. These modifications require extensive future study to reason about how to create aligned AI systems, and answers to these questions are critical in the pathway of human-AI alignment.

7.5 Re-Interpreting Incorrect Reward Functions

Alongside the effort to support humans in writing better reward functions, a complementary approach is to assume their ability to write reward functions will always be imperfect, no matter how good the tooling for specifying, inspecting, and modeling

behaviors. Given this assumption, it is possible to study the types of errors that people make, and extract patterns in those errors—such as our previous observation that people commonly fail to reason about temporal discounting, as discussed in Chapter 3. If we can predict human errors, can we leverage that information to inform beliefs over candidate correct reward functions and better infer an approximation of the human's intended reward function?

This proposed study would build on Inverse Reward Design methods [72]; these methods assume that each candidate reward function a human designs r_i is an *observation* about the correct reward function, based mostly on the "training" environments the human designer has experienced or considered thus far, which may be different from the set of "test" environments the agent will inevitably encounter. In our proposal, where an Inverse Reward Design method is adapted for the task of reconciling common errors in reward design, one of the main challenges is in designing a sensible prior for inference. We would need to modify the model of the human expert to incorporate these known failure proclivities as part of the prior instead of assuming that the human is approximately optimal, as in the original Inverse Reward Design work. As a consequence of this reinterpretation of the reward design problem, the inferred reward functions should better align with humans' true intent.

This proposal invites some criticism, which must be considered and addressed to contribute useful future research. By designing an inference mechanism to address errors in specifications, we necessarily assume that human capabilities are both fixed and stagnant—which is patently false, particularly when one considers that humans learn through experience when designing and interacting with specifications (i.e., as discussed in Chapter 6). Given that humans are forming conceptual models, learning, and adapting all the time, is this approach of modeling errors as a prior viable? A second criticism of this approach is perhaps more fundamental, and is equally true of learned reward functions more generally (Section 7.4). One of the benefits of using hand-designed reward functions is that, at least in principle, these reward functions are designed to be read and understood, and the reward designer can be held somewhat accountable for the consequences of using their reward function. When

instead the reward function is being re-interpreted through Inverse Reward Design, it becomes harder to hold the expert accountable to system failures, as their specification is dynamically interpreted. This can be addressed in part by asking the expert to assess and confirm the proposed changes to their hand-designed reward function, but some of the potential attribution of fault is still lost.

Bibliography

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *NeurIPS*, pages 9505–9515, 2018.

[2] Sonya Alexandrova, Maya Cakmak, Kaijen Hsiao, and Leila Takayama. Robot programming by demonstration with interactive action visualizations. In *Robotics: science and systems*, 2014.

[3] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4):105–120, 2014.

[4] Dan Amir and Ofra Amir. Highlights: Summarizing agent behavior to people. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1168–1176, 2018.

[5] Ofra Amir, Finale Doshi-Velez, and David Sarne. Summarizing agent strategies. *Autonomous Agents and Multi-Agent Systems*, 33(5):628–644, 2019.

[6] Yotam Amitai and Ofra Amir. "I don't think so": Disagreement-based policy summaries for comparing agents. *arXiv preprint arXiv:2102.03064*, 2021.

[7] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

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

[9] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[10] Javier Antorán, Umang Bhatt, Tameem Adel, Adrian Weller, and José Miguel Hernández-Lobato. Getting a clue: A method for explaining uncertainty estimates. *ICLR*, 2021.

[11] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems (RAS)*, 57(5):469–483, 2009.

[12] Sanjeev Arora, Andrej Risteski, and Yi Zhang. Do GANs learn the distribution? some theory and empirics. In *International Conference on Learning Representations*, 2018.

[13] Akanksha Atrey, Kaleigh Clary, and David Jensen. Exploratory not explanatory: Counterfactual analysis of saliency maps for deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[14] Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning from physical human corrections, one feature at a time. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 141–149, 2018.

[15] Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning robot objectives from physical human interaction. In *Conference on Robot Learning*, pages 217–226. PMLR, 2017.

[16] Gagan Bansal, Besmira Nushi, Ece Kamar, Daniel S Weld, Walter S Lasecki, and Eric Horvitz. Updates in human-AI teams: Understanding and addressing the performance/compatibility tradeoff. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2429–2437, 2019.

[17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017.

[18] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *JMLR*, 2018.

[19] Erdem Bıyık, Dylan P Losey, Malayandi Palan, Nicholas C Landolfi, Gleb Shevchuk, and Dorsa Sadigh. Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences. *International Journal of Robotics Research*, 2020.

[20] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *ICML*, pages 1613–1622, 2015.

[21] Andreea Bobu, Andrea Bajcsy, Jaime F Fisac, Sampada Deglurkar, and Anca D Dragan. Quantifying hypothesis space misspecification in learning from human–robot demonstrations and physical corrections. *IEEE Transactions on Robotics (T-RO)*, 36(3):835–854, 2020.

[22] Serena Booth, W Bradley Knox, Julie Shah, Scott Niekum, Peter Stone, and Alessandro Allievi. The perils of trial-and-error reward design: misdesign through overfitting and invalid task specifications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 5, pages 5920–5929, 2023.

[23] Serena Booth, Christian Muise, and Julie Shah. Evaluating the interpretability of the knowledge compilation map: Communicating logical statements effectively. In *IJCAI*, pages 5801–5807, 2019.

[24] Serena Booth, Sanjana Sharma, Sarah Chung, Julie Shah, and Elena L Glassman. Revisiting human-robot teaching and learning through the lens of human concept learning. In *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 147–156. IEEE, 2022.

[25] Serena Booth, Yilun Zhou, Ankit Shah, and Julie Shah. Bayes-TrEx: a Bayesian sampling approach to model transparency by example. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11423–11432, 2021.

[26] Julie Bort. Inside Uber before its self-driving car killed a pedestrian: Sources describe infighting, 'perverse' incentives, and questionable decisions, 2018.

[27] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[28] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of markov chain monte carlo*. CRC press, 2011.

[29] Daniel S Brown, Jordan Schneider, Anca Dragan, and Scott Niekum. Value alignment verification. In *International Conference on Machine Learning*, pages 1105–1115. PMLR, 2021.

[30] Zana Buçinca, Maja Barbara Malaya, and Krzysztof Z Gajos. To trust or to think: cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW1):1–21, 2021.

[31] Thomas J Bussey, MaryKay Orgill, and Kent J Crippen. Variation theory: A theory of learning and a useful theoretical framework for chemical education research. *Chemistry Education Research and Practice*, 14(1):9–22, 2013.

[32] Carrie J Cai, Samantha Winter, David Steiner, Lauren Wilcox, and Michael Terry. "Hello AI": Uncovering the onboarding needs of medical practitioners for human-AI collaborative decision-making. *Proceedings of the ACM on Human-computer Interaction*, 3(CSCW):1–24, 2019.

[33] Thomas Cederborg, Ishaan Grover, Charles L Isbell Jr, and Andrea Lockerd Thomaz. Policy shaping with human teachers. In *IJCAI*, pages 3366–3372, 2015.

[34] Tathagata Chakraborti, Sarath Sreedharan, Sachin Grover, and Subbarao Kambhampati. Plan explanations as model reconciliation–an empirical study. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 258–266. IEEE, 2019.

[35] Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang, and Subbarao Kambhampati. Plan explanations as model reconciliation: Moving beyond explanation as soliloquy. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[36] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. In *NeurIPS*, pages 8928–8939, 2019.

[37] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with AutoRL. *IEEE Robotics and Automation Letters*, 4(2):2007–2014, 2019.

[38] Brian Christian. *The alignment problem: Machine learning and human values*. WW Norton & Company, 2020.

[39] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in neural information processing systems*, pages 4299–4307, 2017.

[40] Benjamin Cohen, Ioan A Şucan, and Sachin Chitta. A generic infrastructure for benchmarking motion planners. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 589–595. IEEE, 2012.

[41] Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.

[42] Yuchen Cui, Qiping Zhang, Alessandro Allievi, Peter Stone, Scott Niekum, and W Bradley Knox. The EMPATHIC framework for task learning from implicit human feedback. In *Conference on Robot Learning*. PMLR, 2020.

[43] Marco Cusumano-Towner and Vikash K Mansinghka. AIDE: an algorithm for measuring the accuracy of probabilistic inference algorithms. In *NeurIPS*, 2017.

[44] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[45] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv*, 2017.

[46] Anca Dragan and Siddhartha Srinivasa. Familiarization to robot motion. In *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*, pages 366–373, 2014.

[47] Anca D Dragan, Kenton CT Lee, and Siddhartha S Srinivasa. Legibility and predictability of robot motion. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 301–308. IEEE, 2013.

[48] Alice Driver, Katherine Elliott, and Andrew Wilson. Variation theory based approaches to teaching subject-specific vocabulary within differing practical subjects. *International Journal for Lesson and Learning Studies*, 2015.

[49] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International conference on learning representations*, 2019.

[50] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[51] Chuchu Fan, Xin Qin, and Jyotirmoy Deshmukh. Parameter searching and partition with probabilistic coverage guarantees. *arXiv preprint arXiv:2004.00279*, 2020.

[52] Aleksandra Faust, Anthony Francis, and Dar Mehta. Evolving rewards to automate reinforcement learning. *arXiv preprint arXiv:1905.07628*, 2019.

[53] Nadia Figueroa. *From High-Level to Low-Level Robot Learning of Complex Tasks: Leveraging Priors, Metrics and Dynamical Systems*. PhD thesis, EPFL, Lausanne, Switzerland, 2019.

[54] Matthew Fontaine and Stefanos Nikolaidis. A quality diversity approach to automatically generating human-robot interaction scenarios in shared autonomy. *Robotics: Science and Systems*, 2021.

[55] Maxwell Forbes, Michael Chung, Maya Cakmak, and Rajesh Rao. Robot programming by demonstration with crowdsourced action fixes. In *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, volume 2, 2014.

[56] Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 63–78, 2019.

[57] Adrian Furnham and Hua Chu Boo. A literature review of the anchoring effect. *The journal of socio-economics*, 40(1):35–42, 2011.

[58] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, pages 1050–1059, 2016.

[59] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[60] D. Gentner, J. Loewenstein, and L. Thompson. Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, 95:393–408, 2003.

[61] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7(2):155–170, 1983.

[62] Dedre Gentner and Linsey A Smith. Analogical learning and reasoning. *The Oxford Handbook of Cognitive Psychology*, 2013.

[63] Amirata Ghorbani, James Wexler, and Been Kim. Automating interpretability: Discovering and testing visual concepts learned by neural networks. *NeurIPS*, 2019.

[64] Mary L Gick and Keith J Holyoak. Schema induction and analogical transfer. *Cognitive psychology*, 15(1):1–38, 1983.

[65] Thomas Krendl Gilbert, Nathan Lambert, Sarah Dean, Tom Zick, Aaron Snoswell, and Soham Mehta. Reward reports for reinforcement learning. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, pages 84–130, 2023.

[66] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, pages 2672–2680, 2014.

[67] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representation (ICLR)*, 2015.

[68] Jackson Gorham and Lester Mackey. Measuring sample quality with Stein's method. In *NeurIPS*, pages 226–234, 2015.

[69] Samuel Greydanus, Anurag Koul, Jonathan Dodge, and Alan Fern. Visualizing and understanding Atari agents. In *International Conference on Machine Learning (ICML)*, pages 1792–1801. PMLR, 2018.

[70] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. *Advances in Neural Information Processing Systems*, 2013.

[71] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *ICML*, 2017.

[72] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. *Advances in neural information processing systems*, 30, 2017.

[73] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. In *Bibliometrika*, pages 97–109. Oxford University Press, 1970.

[74] Kris Hauser. Robust contact generation for robot simulation with unstructured meshes. In *16th International Symposium on Robotics Research (ISRR)*, volume 114, page 357. Springer, 2016.

[75] Kris Hauser and Yilun Zhou. Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space. *IEEE Transactions on Robotics (T-RO)*, 32(6):1431–1443, 2016.

[76] Bradley Hayes and Julie A Shah. Improving robot controller transparency through autonomous policy explanation. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 303–312. IEEE, 2017.

[77] Jerry Zhi-Yang He and Anca D Dragan. Assisted robust reward design. *5th Conference on Robot Learning (CoRL)*, 2021.

[78] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[79] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, pages 6626–6637, 2017.

[80] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. In *The collected works of Wassily Hoeffding*, pages 409–426. Springer, 1994.

[81] Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *JMLR*, 15(1):1593–1623, 2014.

[82] Aspen Hopkins and Serena Booth. Machine learning practices outside Big Tech: How resource constraints challenge responsible development. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pages 134–145, 2021.

[83] Tiffany Horter, Elena L Glassman, Julie Shah, and Serena Booth. Varying how we teach: Adding contrast helps humans learn about robot motions. *HRI Human-Interactive Robot Learning (HIRL) Workshop*, 2023.

[84] Sandy H Huang, Kush Bhatia, Pieter Abbeel, and Anca D Dragan. Establishing appropriate trust via critical states. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3929–3936. IEEE, 2018.

[85] Sandy H Huang, David Held, Pieter Abbeel, and Anca D Dragan. Enabling robots to communicate their objectives. *Autonomous Robots*, 43(2):309–326, 2019.

[86] Lukas Huber, Aude Billard, and Jean-Jacques Slotine. Avoidance of convex and concave obstacles with convergence ensured through contraction. *IEEE Robotics and Automation Letters (RA-L)*, 4(2):1462–1469, 2019.

[87] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in Atari. *Advances in neural information processing systems*, 2018.

[88] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, page 0278364920987859, 2021.

[89] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pages 2107–2116. PMLR, 2018.

[90] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 34(10):1296–1313, 2015.

[91] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):3019–3026, 2020.

[92] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. Citeseer, 2015.

[93] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: a diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

[94] Zoe Juozapaitis, Anurag Koul, Alan Fern, Martin Erwig, and Finale Doshi-Velez. Explainable reinforcement learning via reward decomposition. In *IJCAI/ECAI Workshop on explainable artificial intelligence*, 2019.

[95] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research (IJRR)*, 30(7):846–894, 2011.

[96] Eoin M Kenny and Mark T Keane. On generating plausible counterfactual and semi-factual explanations for deep learning. *arXiv preprint arXiv:2009.06399*, 2020.

[97] Seyed Mohammad Khansari-Zadeh and Aude Billard. A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots (AuRo)*, 32(4):433–454, 2012.

[98] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 500–505. IEEE, 1985.

[99] Junkyung Kim, Matthew Ricci, and Thomas Serre. Not-So-CLEVR: learning same–different relations strains feedforward neural networks. *Interface focus*, 8(4):20180011, 2018.

[100] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. The (un) reliability of saliency methods. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 267–280. Springer, 2019.

[101] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[102] Ryo Kitagawa, Yuyi Liu, and Takayuki Kanda. Human-inspired motion planning for omni-directional social robots. In *Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*, pages 34–42, 2021.

[103] W Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis) design for autonomous driving. *Artificial Intelligence*, 316:103829, 2023.

[104] W Bradley Knox, Brian D Glass, Bradley C Love, W Todd Maddox, and Peter Stone. How humans teach agents. *International Journal of Social Robotics*, 4(4):409–421, 2012.

[105] W. Bradley Knox, Stephane Hatgis-Kessell, Sigurdur Orn Adalgeirsson, Serena Booth, Anca Dragan, Peter Stone, and Scott Niekum. Learning optimal advantage from preferences and mistaking it for reward, 2023.

[106] W Bradley Knox, Stephane Hatgis-Kessell, Serena Booth, Scott Niekum, Peter Stone, and Alessandro Allievi. Models of human preference for learning reward functions. *arXiv preprint arXiv:2206.02231*, 2022.

[107] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16, 2009.

[108] W Bradley Knox and Peter Stone. Framing reinforcement learning from human reward: Reward positivity, temporal discounting, episodicity, and performance. *Artificial Intelligence*, 225:24–50, 2015.

[109] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Training a robot via human feedback: A case study. In *International Conference on Social Robotics*, pages 460–470. Springer, 2013.

[110] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, 2017.

[111] Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of AI ingenuity. *DeepMind Blog*, 2020.

[112] Kenneth J Kurtz, Chun-Hui Miao, and Dedre Gentner. Learning by analogical bootstrapping. *The Journal of the Learning Sciences*, 10(4):417–446, 2001.

[113] Minae Kwon, Sandy H Huang, and Anca D Dragan. Expressing robot incapability. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 87–95, 2018.

[114] Isaac Lage, Daphna Lifschitz, Finale Doshi-Velez, and Ofra Amir. Exploring computational user models for agent policy summarization. In *IJCAI: proceedings of the conference*, volume 28, page 1401. NIH Public Access, 2019.

[115] Fabien Lagriffoul, Neil T Dantam, Caelan Garrett, Aliakbar Akbari, Siddharth Srivastava, and Lydia E Kavraki. Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3765–3772, 2018.

[116] Ho Cheong Lam. Elaborating the concepts of part and whole in variation theory: The case of learning chinese characters. *Scandinavian Journal of Educational Research*, 58(3):337–360, 2014.

[117] Ellen J Langer, Arthur Blank, and Benzion Chanowitz. The mindlessness of ostensibly thoughtful action: The role of "placebic" information in interpersonal interaction. *Journal of personality and social psychology*, 36(6):635, 1978.

[118] Steven M LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[119] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. *Accessed 2021-03-08*, 2010.

[120] John D Lee and Katrina A See. Trust in automation: Designing for appropriate reliance. *Human factors*, 46(1):50–80, 2004.

[121] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*, 2018.

[122] Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: a research direction. *arXiv preprint arXiv:1811.07871*, 2018.

[123] Andre Lemme, Yaron Meirovitch, M Khansari-Zadeh, Tamar Flash, Aude Billard, and Jochen J Steil. Open-source benchmarking for learned reaching motion generation in robotics. *Paladyn, Journal of Behavioral Robotics*, 6(1), 2015.

[124] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[125] Jinying Lin, Qilei Zhang, Randy Gomez, Keisuke Nakamura, Bo He, and Guangliang Li. Human social feedback for efficient interactive reinforcement agent learning. In *29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 706–712. IEEE, 2020.

[126] Lo Mun Ling, Pakey Chik, and Ming Fai Pang. Patterns of variation in teaching the colour of light to primary 3 students. *Instructional Science*, 34(1):1–19, 2006.

[127] Mun Ling Lo. *Variation theory and the improvement of teaching and learning*. Göteborg: Acta Universitatis Gothoburgensis, 2012.

[128] Zachary C Lipton. The mythos of model interpretability. *Queue*, 16(3):31–57, 2018.

[129] Robert Loftin, James MacGlashan, Bei Peng, Matthew Taylor, Michael Littman, Jeff Huang, and David Roberts. A strategy-aware technique for learning behaviors from discrete human feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[130] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jeff Huang, and David L Roberts. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous agents and multi-agent systems*, 30(1):30–59, 2016.

[131] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, pages 4765–4774, 2017.

[132] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *JMLR*, 9(Nov):2579–2605, 2008.

[133] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L Roberts, Matthew E Taylor, and Michael L Littman. Interactive learning from policy-dependent human feedback. In *International Conference on Machine Learning*, pages 2285–2294. PMLR, 2017.

[134] Jeffrey Mahler, Rob Platt, Alberto Rodriguez, Matei Ciocarlie, Aaron Dollar, Renaud Detry, Maximo A Roa, Holly Yanco, Adam Norton, Joe Falco, et al. Guest editorial open discussion of robot grasping benchmarks, protocols, and metrics. *IEEE Transactions on Automation Science and Engineering (T-ASE)*, 15(4):1440–1442, 2018.

[135] Ference Marton. *Necessary conditions of learning*. Routledge, 2014.

[136] Ference Marton and Shirley A Booth. *Learning and awareness*. psychology press, 1997.

[137] Ference Marton, Amy BM Tsui, Pakey PM Chik, Po Yuk Ko, and Mun Ling Lo. *Classroom discourse and the space of learning*. Routledge, 2004.

[138] Michael Mhlolo. The merits of teaching mathematics with variation. *Pythagoras*, 34(2):1–8, 2013.

[139] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[140] Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research (IJRR)*, 37(10):1205–1232, 2018.

[141] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Conference on Fairness, Accountability, and Transparency (FAT*)*, pages 220–229, 2019.

[142] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[143] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[144] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[145] Mark Moll, Ioan A Sucan, and Lydia E Kavraki. Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization. *IEEE Robotics & Automation Magazine (RA-M)*, 22(3):96–102, 2015.

[146] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. PyRobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*, 2019.

[147] Glenford J Myers, Tom Badgett, Todd M Thomas, and Corey Sandler. *The Art of Software Testing*, volume 2. Wiley Online Library, 2004.

[148] Nadim Nachar et al. The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology*, 4(1):13–20, 2008.

[149] Radford M Neal et al. MCMC using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

[150] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[151] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

[152] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1, page 2, 2000.

[153] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436, 2015.

[154] Anh Nguyen, Jason Yosinski, and Jeff Clune. Understanding neural networks via feature visualization: A survey. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019.

[155] Scott Niekum, Andrew G Barto, and Lee Spector. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2(2):83–90, 2010.

[156] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *ICML*, pages 4901–4911, Long Beach, California, USA, 09–15 Jun 2019.

[157] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[158] Matthew L Olson, Roli Khanna, Lawrence Neal, Fuxin Li, and Weng-Keen Wong. Counterfactual state explanations for reinforcement learning agents via generative deep learning. *Artificial Intelligence*, page 103455, 2021.

[159] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[160] Fred GWC Paas and Jeroen JG Van Merriënboer. Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of educational psychology*, 86(1):122, 1994.

[161] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*, 2022.

[162] Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, et al. Automated reinforcement learning (AutoRL): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, 2022.

[163] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 1–8. IEEE, 2018.

[164] Danish Pruthi, Bhuwan Dhingra, Livio Baldini Soares, Michael Collins, Zachary C Lipton, Graham Neubig, and William W Cohen. Evaluating explanations: How much do explanations from the teacher aid students? *arXiv preprint arXiv:2012.00893*, 2020.

[165] Jill L Quilici and Richard E Mayer. Role of examples in how students learn to categorize statistics word problems. *Journal of Educational Psychology*, 88(1):144, 1996.

[166] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

[167] Iyad Rahwan, Manuel Cebrian, Nick Obradovich, Josh Bongard, Jean-François Bonnefon, Cynthia Breazeal, Jacob W Crandall, Nicholas A Christakis, Iain D Couzin, Matthew O Jackson, et al. Machine behaviour. *Nature*, 568(7753):477–486, 2019.

[168] Amir Rasouli and John K Tsotsos. Autonomous vehicles that interact with pedestrians: A survey of theory and practice. *IEEE Transactions on Intelligent Transportation Systems (T-ITS)*, 21(3):900–918, 2019.

[169] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. Riemannian motion policies. *arXiv preprint arXiv:1801.02854*, 2018.

[170] Ellis Ratner, Dylan Hadfield-Menell, and Anca D Dragan. Simplifying reward design through divide-and-conquer. *arXiv preprint arXiv:1806.02501*, 2018.

[171] Siddharth Reddy, Anca Dragan, Sergey Levine, Shane Legg, and Jan Leike. Learning human objectives by evaluating hypothetical behavior. In *International Conference on Machine Learning*, pages 8020–8029. PMLR, 2020.

[172] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?" explaining the predictions of any classifier. In *KDD*, pages 1135–1144, 2016.

[173] Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4902–4912, 2020.

[174] Bethany Rittle-Johnson and Jon R Star. Does comparing solution methods facilitate conceptual and procedural knowledge? an experimental study on learning to solve equations. *Journal of Educational Psychology*, 99(3):561, 2007.

[175] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[176] Stuart Russell. *Human compatible: Artificial intelligence and the problem of control*. Penguin, 2019.

[177] Stuart Russell and Peter Norvig. AI a modern approach. *Learning*, 2(3):4, 2005.

[178] Dorsa Sadigh, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems*, 2017.

[179] Lindsay Sanneman. *Transparent Value Alignment: Foundations for Human-Centered Explainable AI in Alignment*. PhD thesis, Massachusetts Institute of Technology, 2023.

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

[181] Pedro Sequeira and Melinda Gervasio. Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations. *Artificial Intelligence*, 288:103367, 2020.

[182] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[183] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[184] Satinder Singh, Richard L Lewis, and Andrew G Barto. Where do rewards come from. In *Proceedings of the annual conference of the cognitive science society*, pages 2601–2606. Cognitive Science Society, 2009.

[185] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. How can we fool LIME and SHAP? adversarial attacks on post hoc explanation methods. *AAAI Conference on Artificial Intelligence, Ethics, and Society (AIES)*, 2020.

[186] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv:1706.03825*, 2017.

[187] Henry Sowerby, Zhiyuan Zhou, and Michael L Littman. Designing rewards for fast learning. *arXiv preprint arXiv:2205.15400*, 2022.

[188] Jarkko Suhonen, Errol Thompson, Janet Davies, and G Kinshuk. Applications of variation theory in computing education. In *Seventh Baltic Sea Conference on Computing Education Research. Koli, Finland*, 2008.

[189] Harini Suresh, Natalie Lao, and Ilaria Liccardi. Misplaced trust: Measuring the interference of machine learning in human decision-making. In *12th ACM Conference on Web Science*, pages 315–324, 2020.

[190] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[191] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[192] Leila Takayama, Doug Dooley, and Wendy Ju. Expressing thought: improving robot readability with animation principles. In *Proceedings of the 6th international conference on Human-robot interaction*, pages 69–76, 2011.

[193] Benedict Tay, Younbo Jung, and Taezoon Park. When stereotypes meet robots: the double-edge sword of robot gender and personality in human–robot interaction. *Computers in Human Behavior*, 38:75–84, 2014.

[194] Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. In *NeurIPS*, pages 13888–13899, 2019.

[195] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Between-class learning for image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5486–5494, 2018.

[196] Siu Yin Annie Tong. Applying the theory of variation in teaching reading. *Australian Journal of Teacher Education*, 37(10):1, 2012.

[197] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *CVPR*, 2017.

[198] Vaibhav V Unhelkar, Przemyslaw A Lasota, Quirin Tyroller, Rares-Darius Buhai, Laurie Marceau, Barbara Deml, and Julie A Shah. Human-aware robotic assistant for collaborative assembly: Integrating human motion prediction with planning in time. *IEEE Robotics and Automation Letters*, 3(3):2394–2401, 2018.

[199] Julen Urain, Puze Liu, Anqi Li, Carlo D'Eramo, and Jan Peters. Composable Energy Policies for Reactive Motion Generation and Reinforcement Learning. In *Robotics: Science and Systems (RSS)*, Virtual, 2021.

[200] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[201] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A Bayesian approach for policy learning from trajectory preference queries. *Advances in neural information processing systems*, 25:1133–1141, 2012.

[202] Alan Winfield, Eleanor Watson, Takashi Egawa, Emily Barwell, Iain Barclay, Serena Booth, Louise A Dennis, Helen Hastie, Ali Hossaini, Naomi Jacobs, et al. IEEE standard for transparency of autonomous systems. *IEEE*, 2022.

[203] Alan FT Winfield, Serena Booth, Louise A Dennis, Takashi Egawa, Helen Hastie, Naomi Jacobs, Roderick I Muttram, Joanna I Olszewska, Fahimeh Rajabiyazdi, Andreas Theodorou, et al. IEEE P7001: A proposed standard on transparency. *Frontiers in Robotics and AI*, 8:665729, 2021.

[204] Zheng Wu, Wenzhao Lian, Vaibhav Unhelkar, Masayoshi Tomizuka, and Stefan Schaal. Learning dense rewards for contact-rich manipulation tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6214–6221. IEEE, 2021.

[205] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

[206] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*, 2017.

[207] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pages 2174–2182, 2017.

[208] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[209] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding DQNs. In *International Conference on Machine Learning (ICML)*, pages 1899–1908. PMLR, 2016.

[210] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[211] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *ICLR*, 2018.

[212] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado Van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.

[213] Yilun Zhou, Serena Booth, Nadia Figueroa, and Julie Shah. RoCUS: Robot controller understanding via sampling. *Conference on Robot Learning*, 2021.

[214] Yilun Zhou, Serena Booth, Marco Tulio Ribeiro, and Julie Shah. Do feature attribution methods correctly attribute features? *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[215] Yilun Zhou and Kris Hauser. 6DOF grasp planning by optimizing a deep learning scoring function. In *RSS Workshop on Revisiting Contact – Turning a Problem into a Solution*, 2017.

[216] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendix A

Specify

A.1 RL Practitioner Survey

We invited RL practitioners from 2 Fortune 500 company's AI research divisions and from 2 US Research Universities (R1) to participate in this survey. Practitioners were entered into a raffle for a $15 USD gift card in exchange for participating. Practitioners were screened based on whether or not they had designed a reward function in the past year. Only those who affirmed this were able to proceed with the survey. Multiple options may be selected for any given question. 24 practitioners completed the survey. The survey questions are as follows, and the number of respondents for each option are indicated:

- What domain have you designed a reward function for *most recently*? (Mark all that apply, but only for your most recent domain)
  - A gridworld task: 8 respondents
  - A classic RL task – like CartPole or Mountain Car: 2 respondents
  - A robotics task: 12 respondents
  - A multi-agent task (e.g., Hanabi): 5 respondents
  - An Atari game or other game: 5 respondents
  - Other: please specify: 3 respondents: A continuous control task, a racing game, and a command and control task
- How did you write an initial reward function? (Mark all that apply.)
  - By applying intuition and considering how the agent would learn: 15 respondents
  - By embedding domain knowledge of how the agent should behave: 13 respondents
  - Using a reward function someone else had written (e.g., from a publication or shared implementation): 15 respondents

  - By specifying a performance objective for the task: 10 respondents
  - By inverse reinforcement learning, or some other demonstration-based approach: 6 respondents
  - Other: please specify: 1 respondent: By defining a goal region

- Did you shape your reward function?
  - Yes: 15 respondents
  - No: 6 respondents
  - I don't know: 1 respondent
  - Other: please specify: 2 respondents: Tried, but didn't have much success; Using human feedback to shape

- Did you use trial-and-error to refine your reward function?
  - Yes: 22 respondents
  - No: 2 respondents
  - Other: please specify: 0 respondents

- During trial-and-error reward design, how did you evaluate your reward function(s)?
  - By viewing the agent's behavior after it has finished training (and would not be trained further): 18 respondents
  - By viewing the agent's behavior during a training session (that was continued further): 12 respondents
  - By plotting some performance metric against time or learning iterations: 15 respondents
  - By plotting return from the changing reward function against time or learning iterations: 6 respondents
  - By scoring example trajectories: 2 respondents
  - Other: please specify: 0 respondents

- Did you observe suboptimal behavior after training your agent?
  - Yes - I observed the agent taking advantage of a loophole in the reward function, so I changed the function to remove the loophole: 7 respondents
  - Yes - I observed the agent's behavior plateauing at unsatisfactory performance, so I changed the reward function to help it learn beyond the plateau: 13 respondents
  - Yes - I observed the agent's behavior plateauing at unsatisfactory performance, so I changed the learning algorithm or hyperparameter. : 11 respondents
  - No: 4 respondents
  - Other: please specify: 0 respondents

A.2 User Study Details

A.2.1 Expert Participant Recruitment

To recruit study participants, we sent recruitment emails to relevant listservs of computer science researchers and to faculty at four US research universities (R1). These faculty members passed the recruitment email along to their research groups.

A.2.2 User Study Protocol

The next five pages contain a screenshot of the Jupyter notebook used for the first 55 minutes of the study.
The expert participants first read a description of the problem, as follows:

- The goal of the hungry-thirsty domain is to teach an agent to eat as much as possible. There's a catch, though: the agent can only eat when it's not thirsty. Thus, the agent cannot just "hang out" at the food location and keep eating because at some point it will become thirsty and eating will fail.
- The agent always exists for 200 timesteps.
- The grid is 4×4[1]. Food is located in one randomly-selected corner, while water is located in a different (random) corner.
- At each timestep, the agent may take one of the following actions: move (up, down, left, right), eat, or drink.
- But actions can fail:
  - The drink action fails if the agent is not at the water location.
  - The eat action fails if the agent is thirsty, or if the agent is not at the food location.
  - The move action fails if the agent tries to move through one of the red barriers (depicted below).
- If the agent eats, it becomes not-hungry for one timestep.
- If the agent drinks, it becomes not-thirsty.
- When the agent is not-thirsty, it becomes thirsty again with 10% probability on each successive timestep.

The experts then run a cell which shows a GIF of a Hungry Thirsty agent.

The experts are then tasked with specifying the reward function, learning algorithm, and hyperparameters, as shown in the following pages. To avoid biasing

[1]For the first 12 participants, the grid was instead 6×6

participants toward trial-and-error reward design, the order in which they were asked to select the reward function or the algorithm choice was randomized.

For the reward function, the experts set the rewards for each state $\texttt{H} \wedge \texttt{T}, \texttt{H} \wedge \neg\texttt{T}, \neg\texttt{H} \wedge \texttt{T}, \neg\texttt{H} \wedge \neg\texttt{T}$ from the range $[-1, 1]$ in 0.05 increments. The hyperparameters vary slightly across algorithms — for example, DDQN uses an ϵ-greedy action selection strategy, while PPO and A2C instead use an entropy term for exploration. Specifically, PPO and A2C use this entropy term in their loss functions, and this requires the user to set an entropy coefficient for regularization.

After selecting their choice of reward function, learning algorithm, and hyperparameters, the expert can start training the agent. As it trains, the Jupyter notebook plots three graphs. The first is the true metric performance for each episode, which corresponds to the stated goal: "The goal of the hungry-thirsty domain is to teach an agent to eat as much as possible." The second graph corresponds to the undiscounted return for each episode, which is based on the expert's own reward function. The third and final graph consists of a state visitation distribution heatmap, which shows where the agent is spending its time. Darker red means more state visits, and lighter red means fewer visits.

After training an agent (or cutting training short), the expert could then review training in three different ways. They could `select_run_and_show_agent()`, which allows them to see any trained agent's performance for a single, randomly-initialized episode. They could instead `view_training_runs()`, which allows them to review the training graphs for any agent. Or they could `review_past_run()`, which allows them to review their reward function, algorithm choice, and hyperparameter selections for any past agent.

Finally, 55 minutes into the study, experts were asked to submit their final, best configuration. They could choose from any of the configurations they tried during the study.

A.2.3 Follow Up Questions

After 55 minutes, participants submitted their best attempt at training an agent to solve Hungry Thirsty. In the remaining 5 minutes, we asked participants five structured questions:

1. Please describe your approach to training RL agents.
2. How well does this process mimic your past experience of training RL agents?
3. Is there some missing information from this interface which you wished you had access to?
4. Does your submitted agent meet your expectations?
5. Did you shape your reward function?

Hungry-Thirsty Domain

The goal of the hungry-thirsty domain is to teach an agent to eat as much as possible.
There's a catch, though: the agent can only eat when it's not thirsty.
Thus, the agent cannot just “hang out” at the food location and keep eating because at some point it will become thirsty and eating will fail.

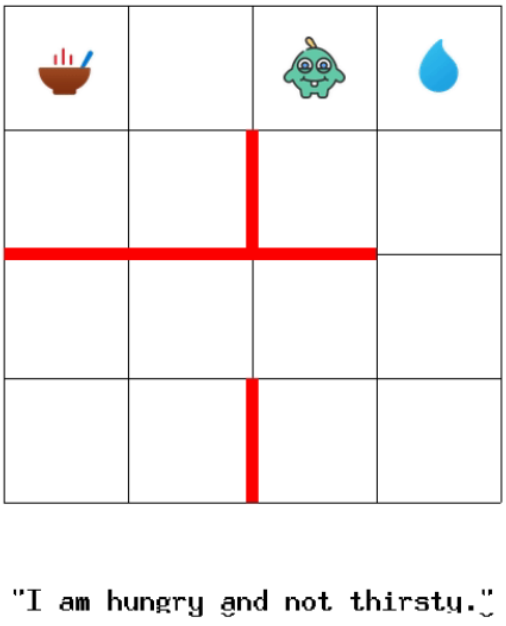
- The agent always exists for 200 timesteps.
- The grid is 4x4. Food is located in one randomly-selected corner, while water is located in a different (random) corner.
- At each timestep, the agent may take one of the following actions: move (up, down, left, right), eat, or drink. But actions can fail:
  - The drink action fails if the agent is not at the water location.
  - The eat action fails if the agent is thirsty, or if the agent is not at the food location.
  - The move action fails if the agent tries to move through one of the red barriers (depicted below).
- If the agent eats, it becomes not-hungry for one timestep.
- If the agent drinks, it becomes not-thirsty.
- When the agent is not-thirsty, it becomes thirsty again with 10% probability on each successive timestep.

See an example of the game below.

In [1]:
```
# import notebooks; do not edit
%run setup_reward_and_learning_alg.ipynb
%run training_and_model_eval.ipynb

# show agent
from IPython.display import Image
Image("../Assets/h-t-small.gif", width=450)
```

Out[1]:



Your Task

We haven't specified the reward function, learning algorithm, or algorithm hyperparameters; you'll need to fill in these details using the code cells below.

Your Study ID

Replace the YOUR_NAME string with your full name to generate a unique study ID.

Note: do *not* re-run the cells after entering data using these Jupyter notebook widgets.

In [2]:
```
set_study_id()
```

Your name: AAAI_2023 Submit

```
Your study ID is: b25b8922b18b04834f444d98afb1d6ea.
```

Design your Reward Function and Select Your RL Algorithm

For your reward function, you need to assign a value r(s) to each state. The state is composed of the thirst and hunger status of the agent. You must assign a value for each state:

- r(Hungry AND Thirsty) = ???; r(Hungry AND Not Thirsty) = ???; r(Not Hungry AND Thirsty) = ???; r(Not Hungry AND Not Thirsty) = ???

For your RL algorithm, you will need to choose your algorithm. Your options are:

- A2C, DDQN, PPO

After choosing your RL algorithm, you will need to select the hyperparameters. If you change your learning algorithm, you may need to re-run the hyperparamater selection (below).

In [3]:
```
# select the learning algorithm and reward function parameters
selectors = reward_and_alg_selector()
selectors
```

Reward for state: hungry AND thirsty -0.50

Reward for state: hungry AND not thirsty -0.25

Reward for state: not hungry AND thirsty 0.75

Reward for state: not hungry AND not thirsty 1.00

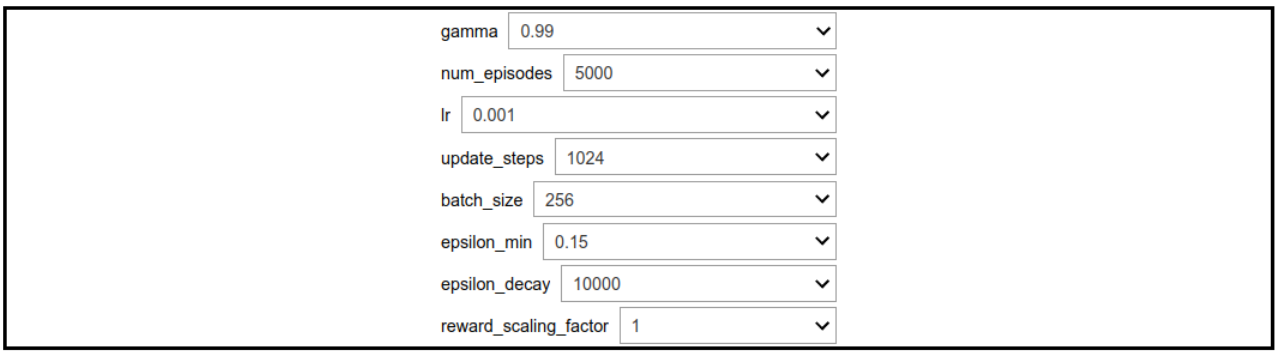
Algorithm Choice DDQN

Hyperparameters

!!! If you change your learning algorithm selection, you will need to rerun the following hyperparameter selection cell as well. !!!

- Hyperparameters Required for All Algorithms:
  - **Gamma**: the discount factor for the environment. A small Gamma means the agent prioritizes only immediate rewards (i.e., the agent is myopic), while a larger Gamma means the agent tends to also consider future rewards.
  - **Num_Episodes**: the number of episodes to train for. A smaller number means the experiments are faster, but contain less experience to learn from.
  - **Learning Rates**: the learning rate is used for training all networks: the Q-network for DDQN, and the actor and critic networks for A2C and PPO. Smaller learning rates make smaller updates to the network weights (and hence optimization is slower), while larger learning rates make larger updates.
  - **reward_scaling_factor**: a mulitplicative factor (σ) applied to the reward function defined above: $r'(s) = \sigma r(s)$. If set to 1, the reward function is unchanged.
- For A2C and PPO:
  - **Entropy_Coeff**: Only applicable to A2C and PPO, this is the entropy regularization coefficient which rewards entropy in the loss function. A smaller value means the loss encourages a less uniform distribution over actions (meaning less exploration, more exploitation).
- For DDQN and PPO:
  - **Update_Steps**: Only applicable to DDQN and PPO, this is the frequency with which to perform updates. A smaller number means more frequent updates, which is slower but more information dense.
- A2C Only:
  - **n_step_update**: How many steps should the agent take before updating the actor-critic network? A smaller number means more frequent updates, which is slower and higher variance. A larger number means less frequent updates, which is faster and lower variance.
- DDQN Only:
  - **Epsilon_Min**: DDQN uses an epsilon-greedy strategy. Epsilon decreases over time to encourage initial exploration (starting at epsilon=1). epsilon_min corresponds to the floor for the epsilon value. A larger epsilon_min means more exploration, less exploitation. A smaller epsilon min means less exploration, more exploitation.
  - **Epsilon_Decay**: Over time, epsilon decreases from 1 to epsilon_min. Every time step, epsilon decreases by 1/epsilon_decay.
  - **Batch_Size**: The number of samples to take from the experience replay buffer from which to calculate the loss and update the deep Q Network. A smaller number is faster to run but contains less experience.
- PPO Only:
  - **Eps_Clip**: In PPO, the estimated advantage function is clipped to handle variance. If the probability ratio between the new policy and the old policy falls outside the range (1 — ε) and (1 + ε), the advantage function is clipped. A smaller eps_clip value is more permissive; a larger eps_clip value is more restrictive and allows for less substantial policy changes.

In [4]:
```
# select the learning algorithm hyperparameters
# !!! If you change the algorithm choice, you will need to re-run this !!!
alg = get_params(widget_ref=selectors)["Algorithm Choice"]
select_learning_alg_params = construct_hyperparam_selector(alg_name = alg)
select_learning_alg_params
```



Training Time!

For evaluating our reward functions, algorithm selection, and hyperparameters, we plot training performance according to *fitness* and *undiscounted return*.

Each episode consists of a trajectory $\tau = [(s_0, a_0, s_1), (s_1, a_1, s_2), \dots]$.

Fitness is computed as the sum of states in which the agent is not hungry:

- Fitness := $\Sigma_{(s,a,s')\in\tau}\ \mathbb{1}$(s[is_hungry] == False)

Undiscounted return is computed using the reward function you specified:

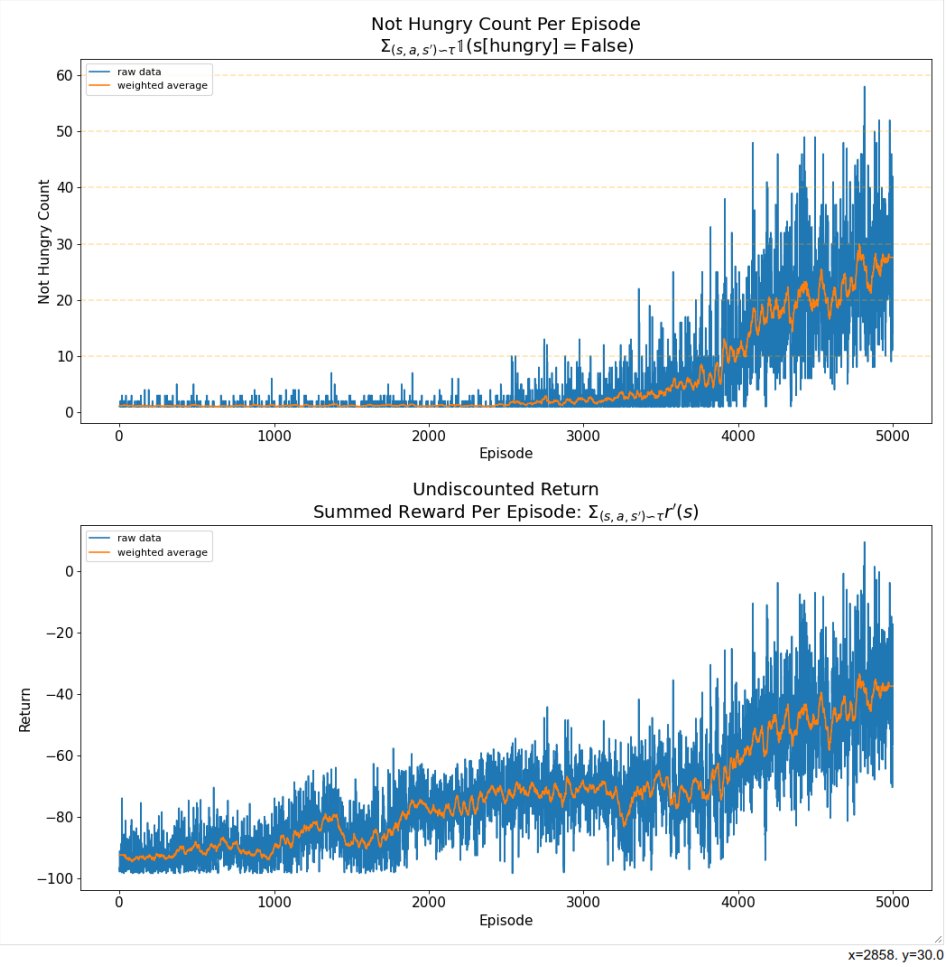
- Undiscounted Return := $\Sigma_{(s,a,s')\in\tau}\sigma r(s) = \Sigma_{(s,a,s')\in\tau} r'(s)$

You may wish to go back and change one or more of your reward function, learning algorithm, or learning algorithm parameters.

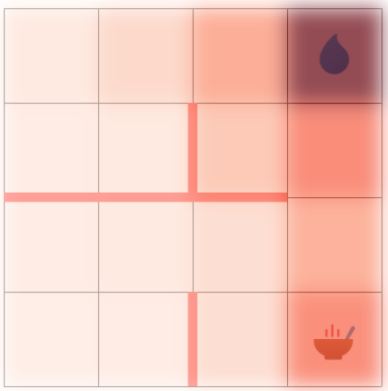
You can cut training off early, but you won't be able to resume training a partially-trained agent.

In [5]:
```
%matplotlib notebook

train_agent(alg_and_reward_params=get_params(widget_ref=selectors),
            hyper_params=get_params(widget_ref=select_learning_alg_params),
            study_id=study_id)
```



x=2858. y=30.0



In [ ]:
```
# view the agent
select_run_and_show_agent()
```

In [ ]:
```
view_training_runs()
```

In [ ]:
```
review_past_run()
```

Final Submission

When you are finished training your agent(s) and choosing which agent is best, run this cell and make your selection.

If the agent you submit is a top-10 performer in this user study, we will award you a $10 bonus after the conclusion of our study.

In [ ]:
```
submit_agent()
```

A.3 Computational Experiments

In this section, we include supporting analysis for the computational experiments. Table A.1, Fig. A-1, and Fig. A-3 correspond to **H1: Reward functions are not universally effective.**

- Table A.1 presents the Hoeffding Bound and Mann Whitney U-test p-values to compare the average cumulative mean performances achieved by policies learned with reward functions across varied hyperparameters (e.g., $\gamma = 0.99$ vs $\gamma = 0.8$).
- Fig. A-1 presents parallel coordinate plots, highlighting the reward functions which resulted in the largest absolute difference in true cumulative performance across varied hyperparameters (e.g., $\gamma = 0.99$ vs $\gamma = 0.8$).
- Fig. A-3 presents parallel coordinate plots, highlighting the reward functions which resulted in the largest absolute difference in true cumulative performance across varied algorithm choices (e.g., PPO vs. DDQN).

Table A.2, Fig. A-2, and Fig. A-4 correspond to **H2: Reward functions are not universally optimal.**

- Table A.2 presents the Hoeffding Bound and Mann Whitney U-test p-values to compare the average cumulative mean performances achieved by the best-performing policies learned with reward functions across varied hyperparameters (e.g., $\gamma = 0.99$ vs $\gamma = 0.8$).
- Fig. A-2 presents parallel coordinate plots, highlighting the reward functions which result in the highest true cumulative performance for each hyperparameter (e.g., $\gamma = 0.99$).
- Fig. A-4 presents parallel coordinate plots, highlighting the reward functions which result in the highest true cumulative performance for each algorithm (e.g., DDQN).

A.4 Deep RL Implementation Details & Hyperparameters

Alongside this paper, we release all of the code for our experiments—including the implementations of the RL algorithms we use: Q-Learning, A2C, DDQN, and PPO. It is known that these algorithms are not only sensitive to hyperparameters, but also potentially to details of the implementation [49]. Unless specified elsewhere in the text, we use the hyperparameters presented in Table A.3.

Table A.1: **H1: Reward functions are not universally effective.** For each large-scale computational comparison experiment (e.g., $\gamma = 0.99$ vs. $\gamma = 0.5$), we find the 3 reward functions which result in the maximal difference in the cumulative true performance metric. To confirm that these differences are not simply simply due to sampling bias, we retrain agents using each of these reward functions 1000 additional times. We then compute the 90% Hoeffding bound as well as a Mann Whitney U-test over this larger set of data. In all cases, we find that the 90% Hoeffding bounds are non-overlapping, and that the difference in underlying distributions are statistically significant. In sum, these reward functions all result in high performance for one experiment variation, and low performance for another variation.

Reward Function	Experiment	Hoeffding Bound	p-value
[-1.0, -1.0, 0.5, -0.5]	$\gamma = 0.99$	[-2,471; 12,798]	< 0.01
	$\gamma = 0.5$	[85,830; 101,099]	
[-0.05, -0.1, 1.0, 0.5]	$\gamma = 0.99$	[91,486; 106,754]	< 0.01
	$\gamma = 0.5$	[2,312; 17,580]	
[-0.5, -0.5, -0.1, -0.05]	$\gamma = 0.99$	[968; 16,237]	< 0.01
	$\gamma = 0.5$	[88,656; 103,925]	
[-1.0, -0.5, 1.0, 0.0]	$\gamma = 0.99$	[13,800; 29,068]	< 0.01
	$\gamma = 0.8$	[45,594; 60,863]	
[-1.0, -1.0, -1.0, 0.1]	$\gamma = 0.99$	[4,965; 20,234]	< 0.01
	$\gamma = 0.8$	[86,653; 101,922]	
[-0.5, -0.5, 0.0, 0.1]	$\gamma = 0.99$	[14837, 30105]	< 0.01
	$\gamma = 0.8$	[88,180; 103,449]	
[-0.5, -0.05, 0.5, 0.5]	$\alpha = 0.25$	[20,034; 35,303]	< 0.01
	$\alpha = 0.05$	[70,224; 85,493]	
[-1.0, -0.1, -0.1, 1.0]	$\alpha = 0.25$	[15,061; 30,330]	< 0.01
	$\alpha = 0.05$	[53,716; 68,985]	
-0.1, -0.5, -1.0, 1.0]	$\alpha = 0.25$	[23,954, 39,223]	< 0.01
	$\alpha = 0.05$	[68,009; 83,278]	

Table A.2: **H2: Reward functions are not universally optimal.** Comparisons of reward function optimality. For each experiment (i.e., $\gamma = 0.99$), we find the 3 best-performing reward functions where each reward function is tested 10 times. As shown in Figure A-2, none of the 'optimal' reward functions are shared across experiment variations. To assess whether these relationships are not simply due to sampling bias, we re-run these experiments with these top reward functions for 1000 trials each, and report the results here for each top-3 reward function combinations. Specifically, we fix an experimental test condition (i.e., $\gamma = 0.99$), and assess whether the top reward function for that experiment (i.e., $[-0.05, -0.05, 0.5, 0.5]$) outperforms the top reward functions for alternative test conditions (i.e., [-1.0, -1.0, 0.0, 1.0], which is optimal for $\gamma = 0.5$). To make this assessment, we compute the 90% Hoeffding Bound of the average cumulative reward. With 90% probability, the true mean lies within this bound. Second, we compute a Mann Whitney U-test to assess whether the distribution of performance means corresponding to the optimal reward function is greater than that of its comparison, and we report the p-values from these tests. In this table, all but two comparisons show statistical significance.

Test	Reward Function	Selection	Hoeffding Bound	p-value
$\gamma = 0.99$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[90442, 105710]	< 0.01
	[-1.0, -1.0, 0.0, 1.0]	#1 for $\gamma = 0.5$	[52436, 67704]	
$\gamma = 0.99$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[90442, 105710]	< 0.01
	[-0.05, -0.05, 0.0, 1.0]	#2 for $\gamma = 0.5$	[85587, 100856]	
$\gamma = 0.99$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[90442, 105710]	< 0.01
	[-1.0, -1.0, 0.1, 0.1]	#3 for $\gamma = 0.5$	[9998, 25266]	
$\gamma = 0.5$	[-1.0, -1.0, 0.0, 1.0]	#1 for $\gamma = 0.5$	[91071, 106339]	0.99
	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[93017, 108286]	
$\gamma = 0.5$	[-1.0, -1.0, 0.0, 1.0]	#1 for $\gamma = 0.5$	[91071, 106339]	< 0.01
	[-0.05, -0.1, 1.0, 0.5]	#2 for $\gamma = 0.99$	[2311, 17580]	
$\gamma = 0.5$	[-1.0, -1.0, 0.0, 1.0]	#1 for $\gamma = 0.5$	[91071, 106339]	< 0.01
	[-0.5, -0.1, 1.0, 0.5]	#3 for $\gamma = 0.99$	[15209, 30478]	
$\gamma = 0.99$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[90442, 105710]	< 0.01
	[-0.05, 0.0, 0.05, 0.5]	#1 for $\gamma = 0.8$	[73526, 88795]	
$\gamma = 0.99$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[90442, 105710]	< 0.01
	[-0.05, 0.0, 0.5, 0.5]	#2 for $\gamma = 0.8$	[77842, 93111]	
$\gamma = 0.99$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[90442, 105710]	< 0.01
	[-1.0, -1.0, 0.1, 1.0]	#3 for $\gamma = 0.8$	[53252, 68520]	
$\gamma = 0.8$	[-0.05, 0.0, 0.05, 0.5]	#1 for $\gamma = 0.8$	[80029, 95298]	1.0
	[-0.05, -0.05, 0.5, 0.5]	#1 for $\gamma = 0.99$	[99649, 114917]	
$\gamma = 0.8$	[-0.05, 0.0, 0.05, 0.5]	#1 for $\gamma = 0.8$	[80029, 95298]	< 0.01
	[-0.05, -0.1, 1.0, 0.5]	#2 for $\gamma = 0.99$	[73935, 89204]	
$\gamma = 0.8$	[-0.05, 0.0, 0.05, 0.5]	#1 for $\gamma = 0.8$	[80029, 95298]	< 0.01
	[-0.5, -0.1, 1.0, 0.5]	#3 for $\gamma = 0.99$	[56372, 71641]	
$\alpha = 0.25$	[-0.1, -0.1, 1.0, 0.05]	#1 for $\alpha = 0.25$	[69028, 84297]	< 0.01
	[-0.05, -0.05, 0.5, 0.5]	#1 for $\alpha = 0.05$	[46150, 61419]	
$\alpha = 0.25$	[-0.1, -0.1, 1.0, 0.05]	#1 for $\alpha = 0.25$	[69028, 84297]	< 0.01
	[-0.05, -0.1, 1.0, 0.5]	#2 for $\alpha = 0.05$	[53966, 69235]	
$\alpha = 0.25$	[-0.1, -0.1, 1.0, 0.05]	#1 for $\alpha = 0.25$	[69028, 84297]	< 0.01
	[-0.5, -0.1, 1.0, 0.5]	#3 for $\alpha = 0.05$	[29049, 44318]	
$\alpha = 0.05$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\alpha = 0.05$	[90442, 105710]	0.01
	[-0.1, -0.1, 1.0, 0.05]	#1 for $\alpha = 0.25$	[79718, 94987]	
$\alpha = 0.05$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\alpha = 0.05$	[90442, 105710]	< 0.01
	[-0.05, 0.0, 1.0, 0.1]	#2 for $\alpha = 0.25$	[69325, 84594]	
$\alpha = 0.05$	[-0.05, -0.05, 0.5, 0.5]	#1 for $\alpha = 0.05$	[90442, 105710]	< 0.01
	[-0.05, -0.05, 0.5, 0.0]	#3 for $\alpha = 0.25$	[64327, 79596]	

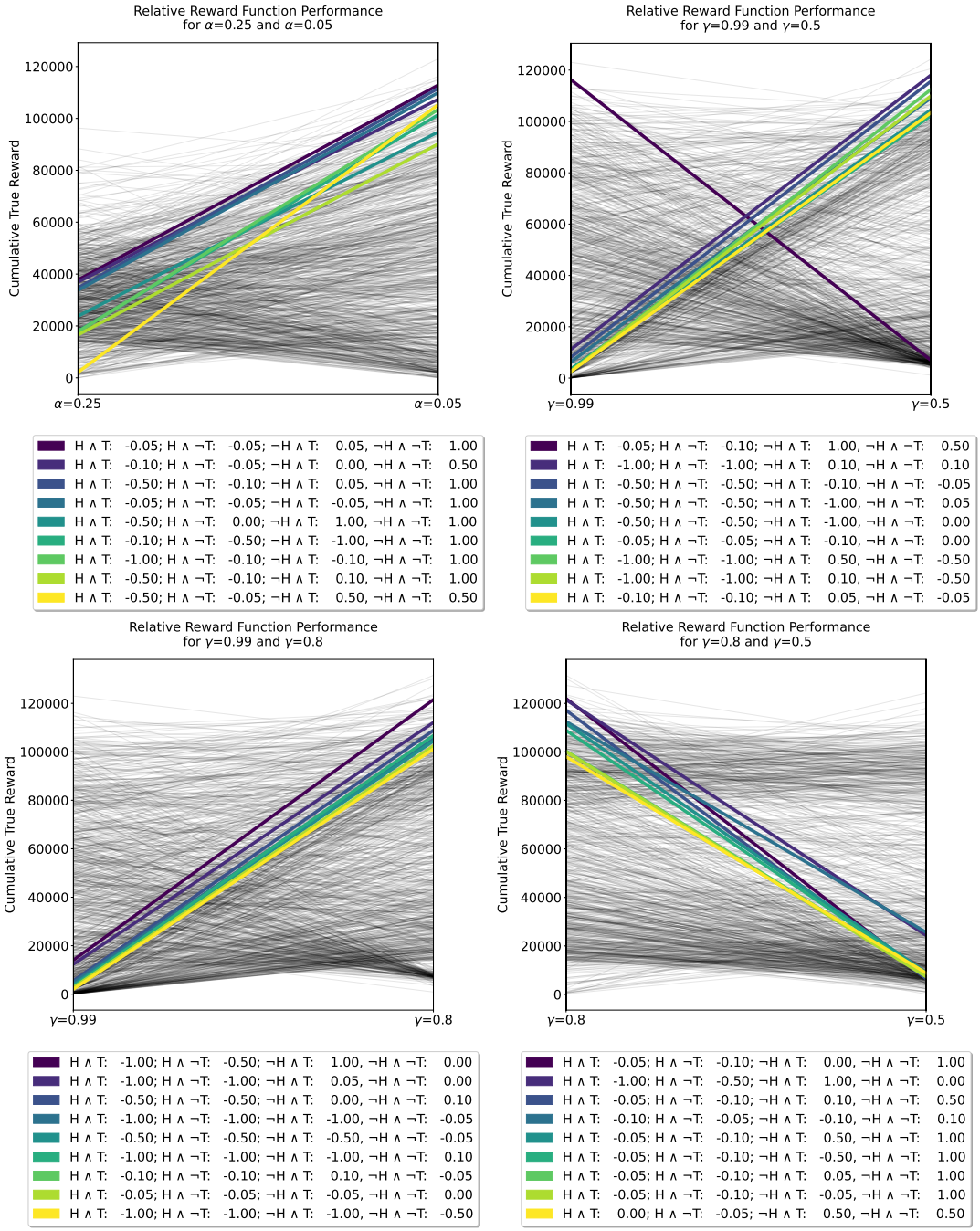


Figure A-1: Parallel coordinate plots which correspond to **H1: Reward functions are not universally effective.** This figure shows plots for $\alpha = 0.25$ vs. $\alpha = 0.05$, $\gamma = 0.99$ vs. $\gamma = 0.5$, and $\gamma = 0.99$ vs. $\gamma = 0.8$. Each line represents a reward function's performance, as measured by the true cumulative performance metric achieved by an average policy trained with the hyperparameter shown on the $x-$axis. The reward functions which result in the highest absolute difference in performance are highlighted. For example, $[-0.05, -0.05, 0.05, 1.0]$ resulted in a cumulative performance of approximately $40,000$ when $\alpha = 0.25$, but instead resulted in a cumulative performance of approximately $100,000$ when $\alpha = 0.05$.

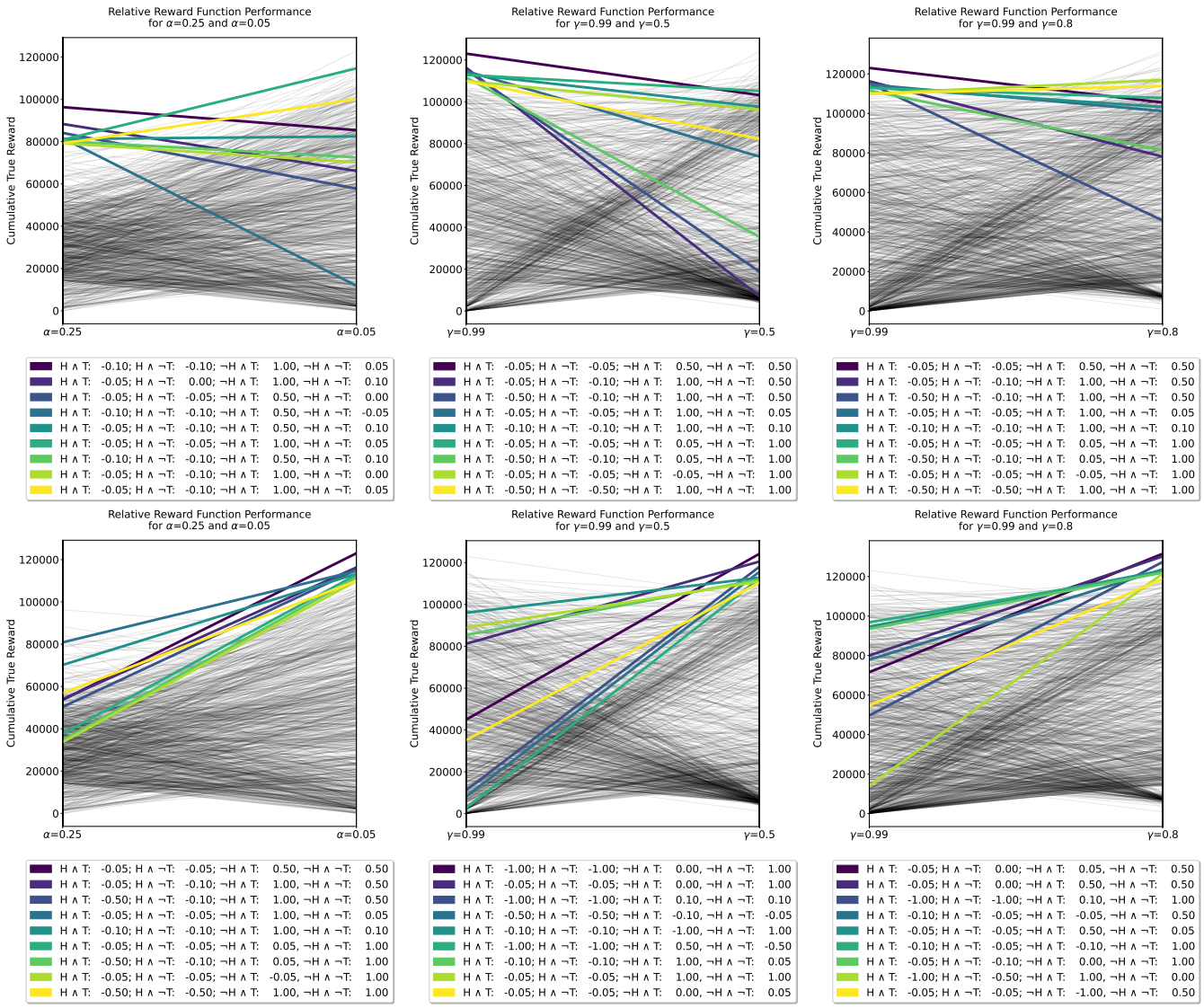


Figure A-2: Parallel coordinate plots which correspond to **H2: Reward functions are not universally optimal.** This figure shows plots for $\alpha = 0.25$ vs. $\alpha = 0.05$, $\gamma = 0.99$ vs. $\gamma = 0.5$, and $\gamma = 0.99$ vs. $\gamma = 0.8$. Each line represents a reward function's performance, as measured by the true cumulative performance metric achieved by an average policy trained with the hyperparameter shown on the $x-$axis. In the top row, the reward functions which result in the best cumulative performance for the first condition—i.e., $\alpha = 0.25$, $\gamma = 0.99$, and $\gamma = 0.99$ respectively—are highlighted. In the bottom row, the reward functions which result in the best cumulative performance for the second condition—i.e., $\alpha = 0.05$, $\gamma = 0.5$, and $\gamma = 0.8$ respectively—are highlighted.

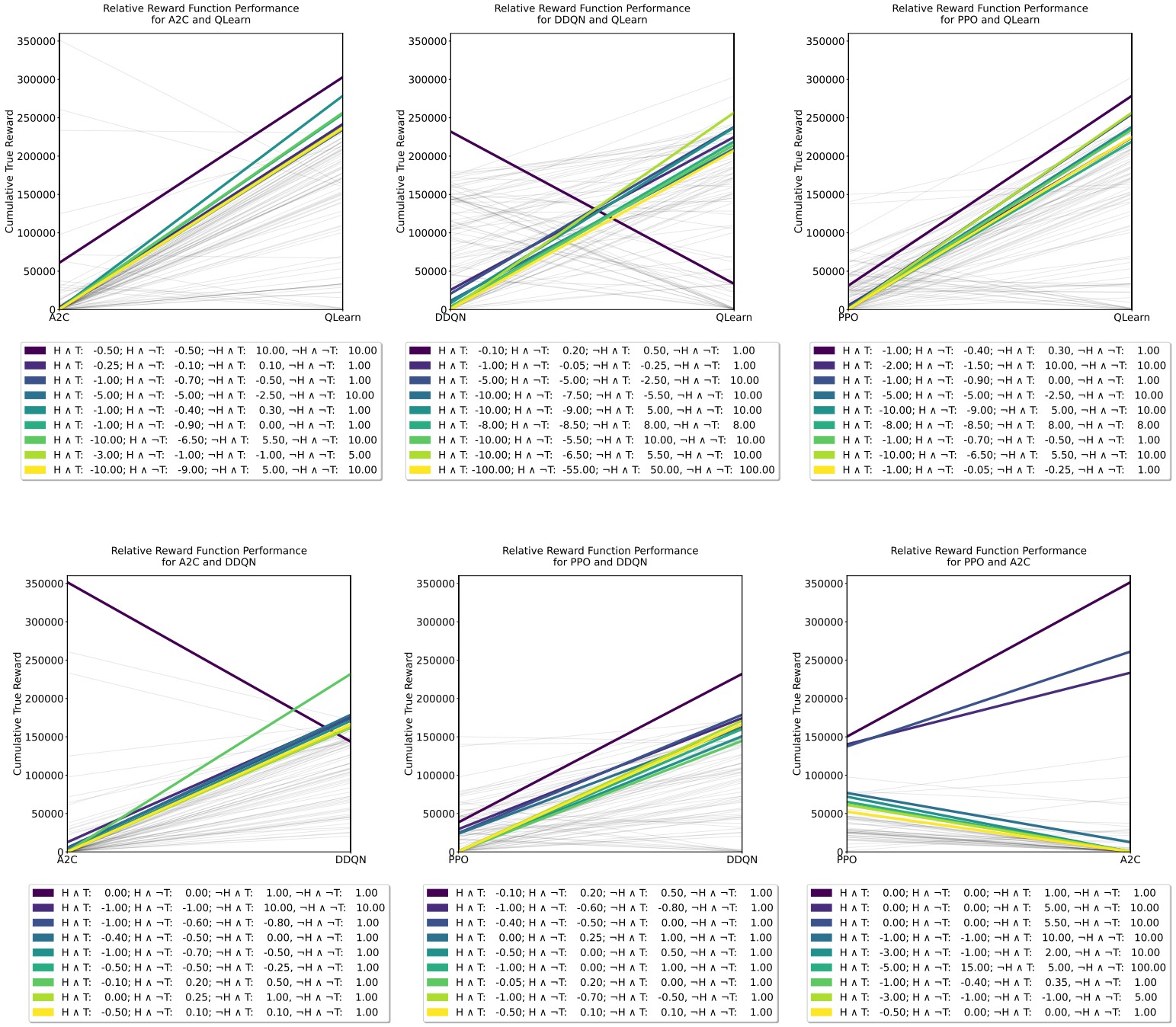


Figure A-3: Parallel coordinate plots which correspond to **H1: Reward functions are not universally effective.** This figure shows plots for A2C vs. Q-learning, DDQN vs. Q-learning, PPO vs. Q-learning, A2C vs. DDQN, PPO vs. DDQN, and PPO vs. A2C. Each line represents a reward function's performance, as measured by the true cumulative performance metric achieved by an average policy trained with algorithm shown on the $x-$axis, using the standard hyperparameters as described in reward-design-Apdx. Section A.4. The reward functions which result in the highest absolute difference in performance are highlighted. For example, $[-0.50, -0.50, 10.00, 10.00]$ resulted in a cumulative performance of approximately $60,000$ when trained with A2C, but instead resulted in a cumulative performance of approximately $300,000$ with Q-learning. Note that these algorithms are each trained for 5000 episodes (instead of the 2000 used for comparing performance across hyperparameter changes).

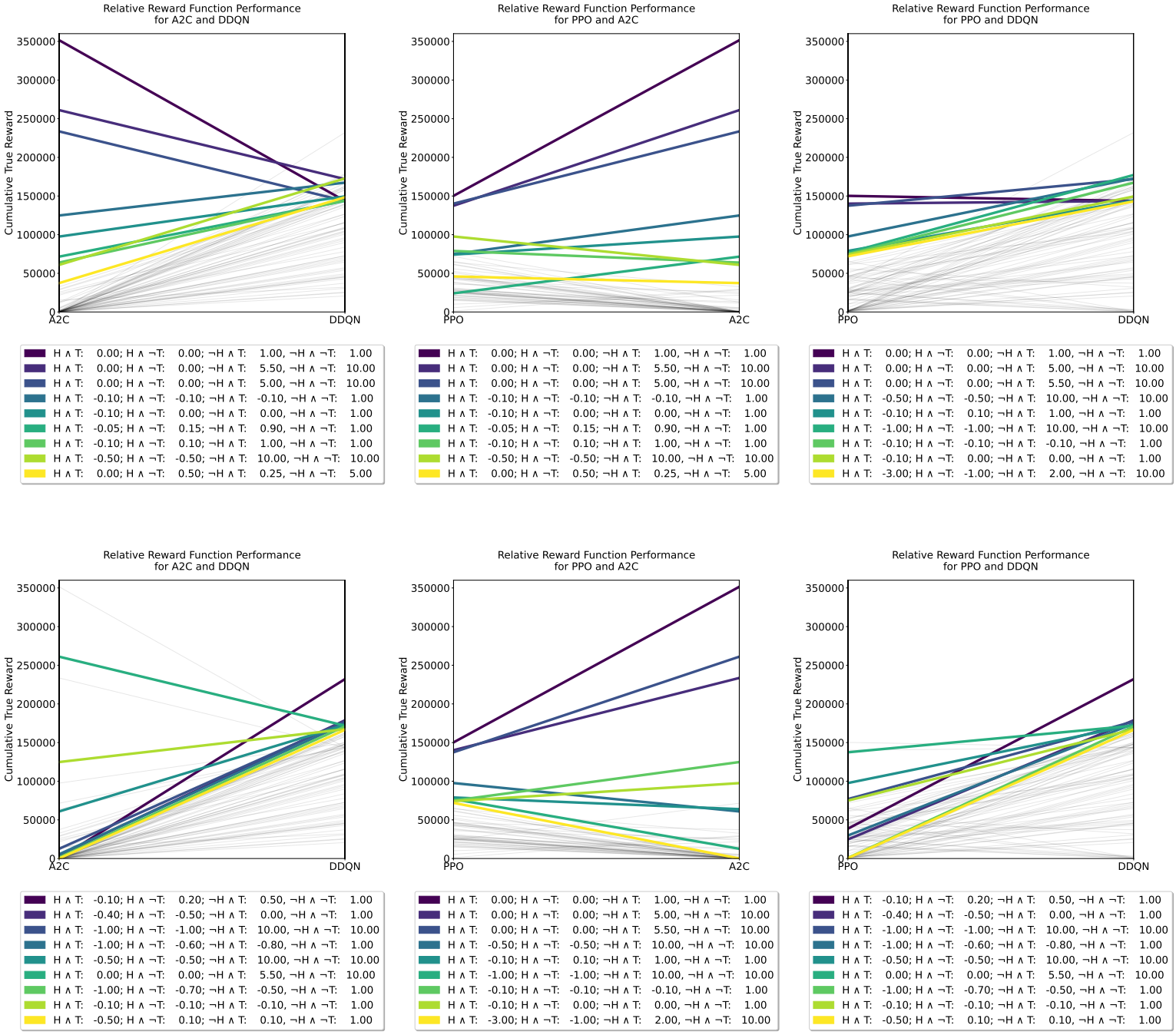


Figure A-4: Parallel coordinate plots which correspond to **H2: Reward functions are not universally optimal.** This figure shows plots for A2C vs. DDQN, PPO vs. A2C, and PPO vs. DDQN. Each line represents a reward function's performance, as measured by the true cumulative performance metric achieved by an average policy trained with the algorithm shown on the $x-$axis. In the top row, the reward functions which result in the best cumulative performance for the first condition—i.e., A2C, PPO, and PPO respectively—are highlighted. In the bottom row, the reward functions which result in the best cumulative performance for the second condition—i.e., DDQN, A2C, and DDQN respectively—are highlighted. The best-performing reward functions are different for A2C vs. DDQN and PPO vs. DDQN, but not for PPO vs. A2C. In this last case, the optimal reward function corresponds to the sparse reward function $[0, 0, 1, 1]$.

Table A.3: Summary of the standard hyperparameters and related implementation details used with each RL algorithm.

Standard Hyperparameter Value	Q-Learning	A2C	DDQN	PPO
γ, discount factor	0.99	0.99	0.99	0.99
α, learning rate	0.05	0.001	0.001	0.005
number of training episodes	2000	5000	5000	5000
neural net hidden layer size	—	144	144	144
neural net structure	—	Actor/Critic	Q-Network/Q-Network	Actor/Critic
neural net activation function	—	ReLU	ReLU	ReLU
entropy coefficient	—	0.01	—	0.01
ϵ-greedy coefficient	0.15	—	0.15	—
ϵ-decay rate	—	—	10000	—
n-step network updates	—	20	—	—
experience replay buffer size	—	—	5000	—
update steps	—	—	128	800
batch size	—	—	128	80
ϵ clipping coefficient (trust region)	—	—	—	0.2

A.5 User Study Overfitting

Say a user selects reward function r_i, but tested several other reward functions, $r_1, r_2, \ldots, r_n$. In the main text, we assess whether users overfit their reward functions to their selected algorithms and hyperparameters by assessing whether *any* of the user's tested reward functions r_j, where $j \in [1, n]$ and $r_i \neq r_j$, outperformed the user's final reward function significantly using *any* of the studied algorithms (DDQN, PPO, and A2C with fixed hyperparameters). Here we propose an alternative metric for overfitting in this context. If the *average* performance of the user's selected reward function over all three tested algorithms is significantly worse than the *average* performance of one of the user's alternative reward functions, we claim it is overfit. As in the former evaluation, we again define the performance difference threshold to be 20000, accumulated over 5000 training episodes and averaged over 10 trials.

Using this alternative metric, we find 11 of 24 users (46%) overfit their reward functions. Note that 6 users are excluded from this evaluation, as their final reward functions were invalid even in the best-case environment configuration. For example, user **P28** submitted the reward function $[-1.0, -0.05, -0.25, 1.0]$, which achieved an average cumulative performance of 8793 across the implementations of DDQN, PPO, and A2C. This same user tested but did not select the alternative reward function $[-1.0, -0.1, 0.0, 1.0]$, which achieved an average cumulative performance of 35367 across the three tested algorithms. Since the performance difference is more than the selected threshold of 20000, we say this user overfit their reward function.

A.6 Compute

We ran these experiments on a combination of local compute and cloud services. Our local compute consists of a machine with 64GB DDR4, a 12-core AMD Ryzen 9 5900 CPU, and a single NVIDIA GeForce RTX 3080 Ti GPU. We used a cloud service which has 480 nodes with Intel Xeon Platinum 8260 CPUs and 4 GB of RAM per core, with no GPUs available. All computers use Ubuntu as the OS. The description of all packages and the specific versions needed to run the code (e.g., NumPy) is included in the released codebase.

A.6.1 Deep RL Agents: Time to Train

Without any parallelization, training a DDQN agent for 5000 episodes on local compute with the standard hyperparameters took on average 186 seconds; training a PPO agent took 415 seconds; and, finally, training an A2C agent took 511 seconds.

Appendix B

Inspect

B.1 Network Architecture for MNIST & Fashion-MNIST

For all experiments on MNIST and Fashion-MNIST, the VAE architecture is shown in Table B.1 (left), and the GAN architecture is shown in Table B.1 (right). For all experiments on MNIST and Fashion-MNIST except for the domain adaptation analysis, the classifier architecture is shown in Table B.2 (left). The classifier used in the domain adaptation analysis is the LeNet architecture, following the original source code released by the authors, shown in Table B.2 (right). VAEs and GANs are trained with binary cross entropy loss. Classifiers are trained with negative log likelihood loss.

Table B.1: Left: VAE architecture; right: GAN architecture.

Encoder input: $28 \times 28 \times 1$
Flatten
Fully-connected 784×400
ReLU
Mean: Fully-connected 400×5
Log-variance: Fully-connected 400×5
Decoder input: 5 (latent dimension)
Fully-connected 5×400
ReLU
Fully-connected 400×784
Reshape $28 \times 28 \times 1$
Sigmoid

Input: 5 (latent dimension)
Reshape $1 \times 1 \times 5$
Conv-transpose: 512 filters, size=4×4, stride = 1
Batch-norm, ReLU
Conv-transpose: 256 filters, size=4×4, stride = 2
Batch-norm, ReLU
Conv-transpose: 128 filters, size=4×4, stride = 2
Batch-norm, ReLU
Conv-transpose: 64 filters, size=4×4, stride = 2
Batch-norm, ReLU
Conv-transpose: 1 filters, size=1×1, stride = 1
Sigmoid

Table B.2: Left: classifier architecture in all experiments except domain adaptation analysis; right: LeNet classifier architecture in domain adaptation analysis (used in code released by ADDA authors).

Input: $28 \times 28 \times 1$
Conv: 32 filters, size $= 3 \times 3$, stride $= 1$
ReLU
Conv: 64 filters, size $= 3 \times 3$, stride $= 1$
Drop-out, prob $= 0.25$
Max-pool, size $= 2 \times 2$
Flatten
Fully-connected 9216×128
ReLU
Drop-out, prob $= 0.5$
Fully-connected 128×10
Soft-max

Input: $28 \times 28 \times 1$
Conv: 20 filters, size $= 5 \times 5$, stride $= 1$
ReLU
Max-pool, size $= 2 \times 2$
Conv: 50 filters, size $= 5 \times 5$, stride $= 1$
ReLU
Max-pool, size $= 2 \times 2$
Flatten
Fully-connected 800×500
ReLU
Fully-connected 500×10
Soft-max

B.2 Fréchet Inception Distance (FID) for VAE and GAN

Table B.3 extends Table 4.1 in Sec. 4.4.2 and lists the FID scores for all VAE and GAN models that we use. These FID scores reveal the GANs are better approximations of the underlying data distributions. Models trained on "all" data are used for high confidence, ambiguous confidence, confidence interpolation and domain adaptation settings. Models trained on data "without [class]" are used for high-confidence failure settings. Models trained on select classes ({2, 4, 5, 7, 8} and {0, 1, 4, 7, 8}) are used for the novel class extrapolation settings.

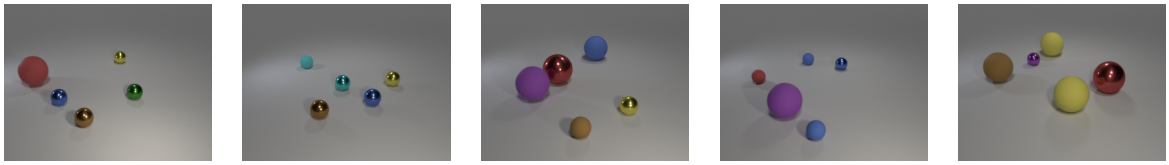
Table B.3: Fréchet Inception Distance (FID) scores for all learned data distributions; a lower score indicates a better distribution fit. Results are computed across 1000 samples. Classes 0 to 9 for Fashion-MNIST correspond to 0: T-shirt, 1: Trouser, 2: Pullover, 3: Dress, 4: Coat, 5: Sandal, 6: Shirt, 7: Sneaker, 8: Bag, and 9: Ankle boot.

Model	Dataset	Data Source	FID
GAN	MNIST	All	11.83
		Without 0	12.10
		Without 1	12.08
		Without 2	13.57
		Without 3	12.71
		Without 4	12.25
		Without 5	12.21
		Without 6	11.86
		Without 7	11.64
		Without 8	12.31
		Without 9	12.34
		{2, 4, 5, 7, 8}	13.45
	Fashion	All	29.44
		Without 0	28.91
		Without 1	31.18
		Without 2	30.11
		Without 3	28.95
		Without 4	30.43
		Without 5	27.67
		Without 6	29.68
		Without 7	28.56
		Without 8	30.87
		Without 9	29.22
		{0, 1, 4, 7, 8}	33.11

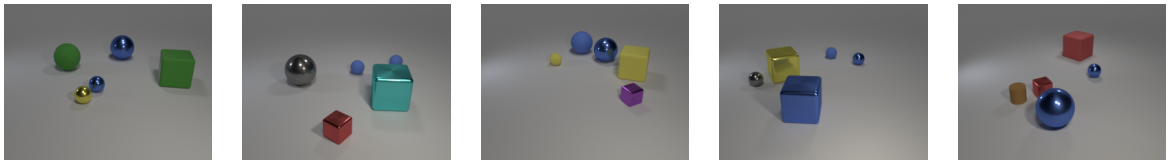
Model	Dataset	Data Source	FID
VAE	MNIST	All	72.33
		Without 0	71.28
		Without 1	75.36
		Without 2	64.77
		Without 3	63.66
		Without 4	66.96
		Without 5	63.31
		Without 6	67.64
		Without 7	62.45
		Without 8	64.14
		Without 9	66.57
		—	—
	Fashion	All	87.89
		Without 0	89.21
		Without 1	92.02
		Without 2	91.20
		Without 3	85.51
		Without 4	88.38
		Without 5	84.17
		Without 6	85.58
		Without 7	84.93
		Without 8	83.66
		Without 9	81.48
		—	—

B.3 High-Confidence Examples

Figure B-1 presents additional high-confidence CLEVR examples and the classifier's predictions.



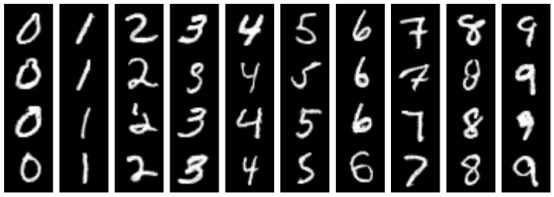
(a) $P_{5\ \text{Sph.}} = 94.8\%$ (b) $P_{5\ \text{Sph.}} = 94.5\%$ (c) $P_{5\ \text{Sph.}} = 94.6\%$ (d) $P_{5\ \text{Sph.}} = 95.2\%$ (e) $P_{5\ \text{Sph.}} = 92.0\%$



(f) $P_{2\ \text{Blue}} = 96.3\%$ (g) $P_{2\ \text{Blue}} = 96.1\%$ (h) $P_{2\ \text{Blue}} = 94.9\%$ (i) $P_{2\ \text{Blue}} = 96.8\%$ (j) $P_{2\ \text{Blue}} = 97.8\%$

Figure B-1: Above, B-1(a)–B-1(e): selected examples classified as containing 5 spheres with high confidence. Below, B-1(f)–B-1(j): selected examples classified as containing 2 blue spheres with high confidence.

Figure B-2 presents additional high-confidence examples for MNIST and Fashion-MNIST.



(a) MNIST

(b) Fashion-MNIST

Figure B-2: High-confidence examples from MNIST and Fashion-MNIST. There are no misclassifications. MNIST columns represent digit 0 to 9, respectively. Fashion-MNIST columns represent T-shirt, trousers, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot, respectively.

B.4 Ambiguous Confidence Examples

Figure B-3 presents additional visualizations for two pairs, Digit 1 vs. Digit 7 from MNIST and T-shirt vs. Pullover from Fashion-MNIST. The confidence plots in the middle confirm that the neural network is indeed making the ambiguous predictions. The t-SNE [132] latent space visualizations at the bottom indicate that the samples lie around the class boundaries and are also in-distribution (i.e., having close proximity to those sampled from the prior).

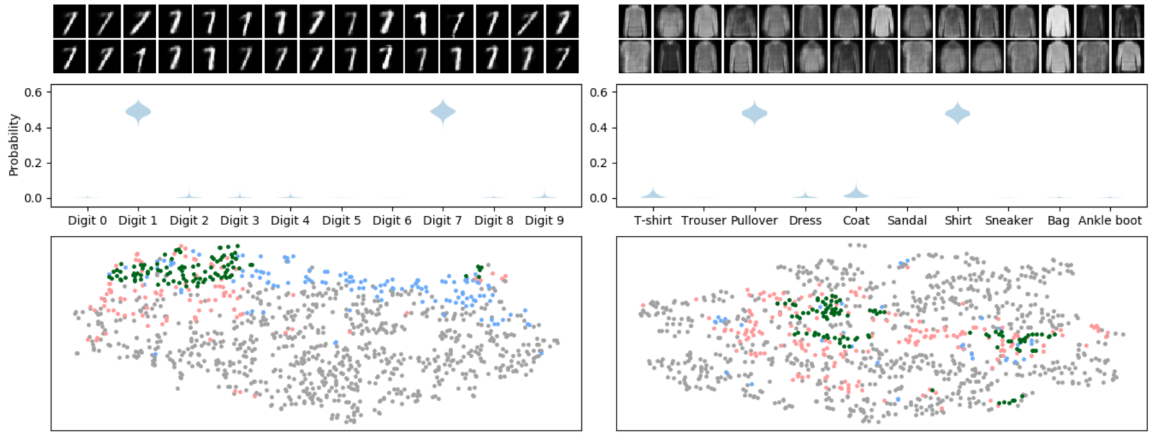


Figure B-3: Left: ambiguous samples for digit 1 vs. 7 in MNIST. Right: ambiguous samples for pullover vs. shirt in Fashion-MNIST. Top: 30 sampled images. Middle: classifier confidence plots on the samples. Bottom: t-SNE latent space visualization: green dots represent ambiguous samples from the posterior, red and blue dots represents samples from the prior that are predicted by the classifier to be either class of interest, and gray dots represents other samples from the prior. The ambiguous samples are on the class boundaries.

In addition, we also sampled for uniformly ambiguous examples (i.e. images that receive around 10% confidence for each class) using the following formulation:

$$u|\vec{x} \sim \text{No}(\max_i f(\vec{x})_i - \min_j f(\vec{x})_j, \sigma^2), \tag{B.1}$$

$$u^* = 0. \tag{B.2}$$

Fig. B-4 shows these samples and their confidence plot.

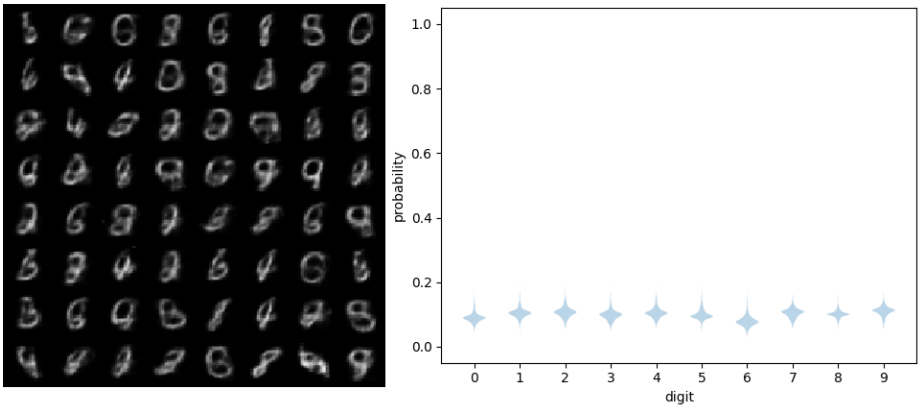


Figure B-4: Uniformly ambiguous images and the confidence plot.

B.5 Ambiguous Confidence with GAN and Modified Classifier

Fig. B-5 shows the ambiguous confidence samples for 0v1, 1v2, ..., 9v0 using the GAN-learned distribution when the classifier is trained with the custom KL loss described in Eq. 4.13.

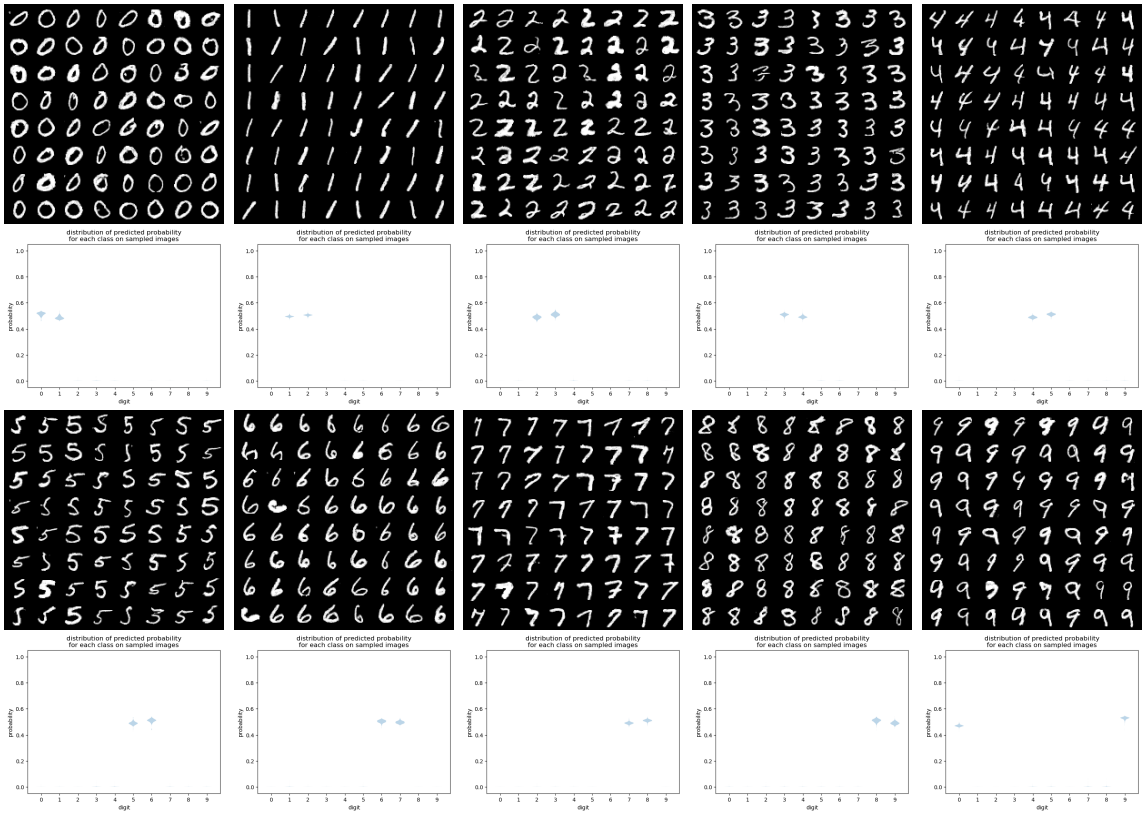
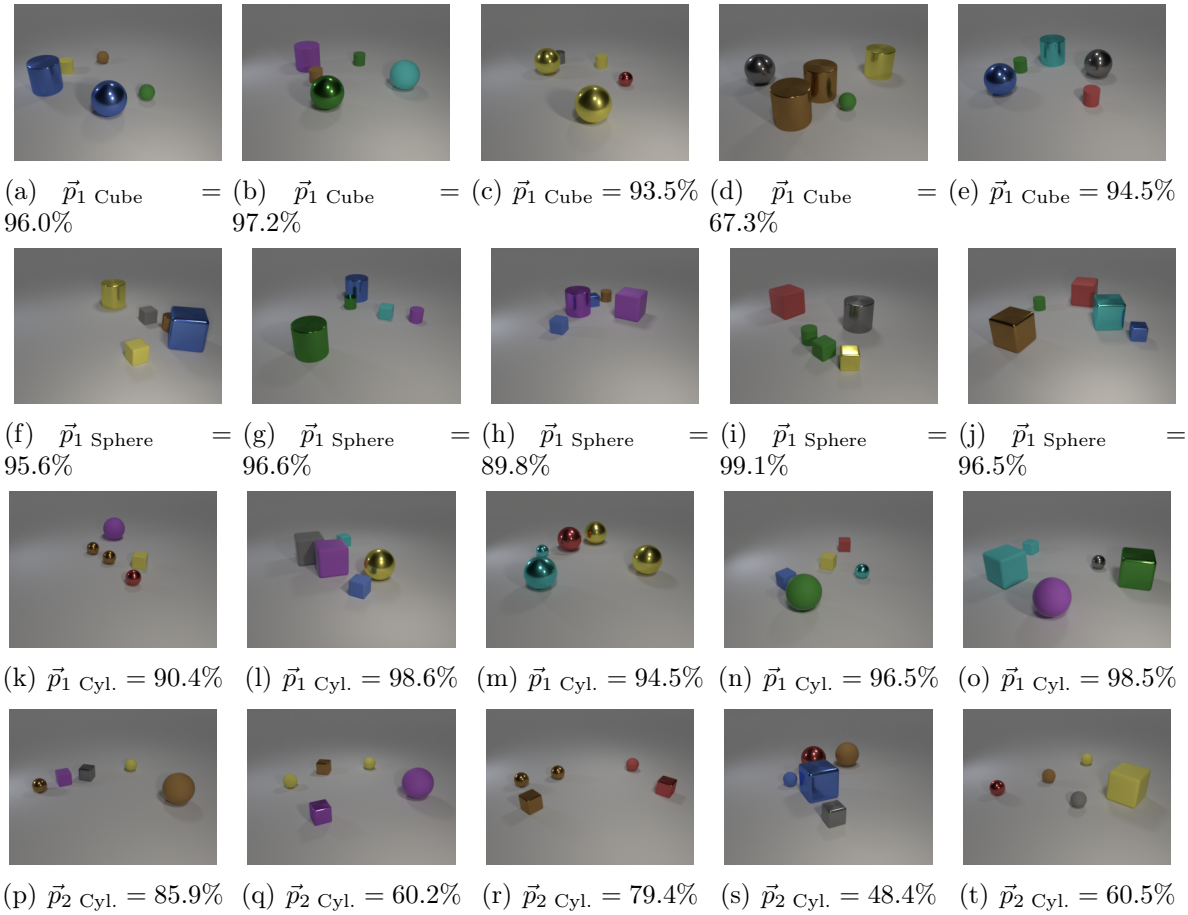


Figure B-5: Sampling results with an explicitly ambivalent classifier and a GAN-learned distribution. Top 2 rows: digit i vs. $i+1$ for $i \in \{0, 1, 2, 3, 4\}$. Bottom 2 rows: digit i vs. $i+1 \pmod{10}$ for $i \in \{5, 6, 7, 8, 9\}$.

B.6 High-Confidence Failure Analysis

Fig. B-6 shows such examples for CLEVR. For each target inference (e.g. "1 Cube"), we exclude objects belonging to the target class from the data distribution.



(a) $\vec{p}_{1\ \text{Cube}} = 96.0\%$ (b) $\vec{p}_{1\ \text{Cube}} = 97.2\%$ (c) $\vec{p}_{1\ \text{Cube}} = 93.5\%$ (d) $\vec{p}_{1\ \text{Cube}} = 67.3\%$ (e) $\vec{p}_{1\ \text{Cube}} = 94.5\%$

(f) $\vec{p}_{1\ \text{Sphere}} = 95.6\%$ (g) $\vec{p}_{1\ \text{Sphere}} = 96.6\%$ (h) $\vec{p}_{1\ \text{Sphere}} = 89.8\%$ (i) $\vec{p}_{1\ \text{Sphere}} = 99.1\%$ (j) $\vec{p}_{1\ \text{Sphere}} = 96.5\%$

(k) $\vec{p}_{1\ \text{Cyl.}} = 90.4\%$ (l) $\vec{p}_{1\ \text{Cyl.}} = 98.6\%$ (m) $\vec{p}_{1\ \text{Cyl.}} = 94.5\%$ (n) $\vec{p}_{1\ \text{Cyl.}} = 96.5\%$ (o) $\vec{p}_{1\ \text{Cyl.}} = 98.5\%$

(p) $\vec{p}_{2\ \text{Cyl.}} = 85.9\%$ (q) $\vec{p}_{2\ \text{Cyl.}} = 60.2\%$ (r) $\vec{p}_{2\ \text{Cyl.}} = 79.4\%$ (s) $\vec{p}_{2\ \text{Cyl.}} = 48.4\%$ (t) $\vec{p}_{2\ \text{Cyl.}} = 60.5\%$

Figure B-6: Sampled high confidence misclassified examples and their associated prediction confidences. For each target constraint (e.g., "1 Cube"), objects from the target class (e.g., cubes) are excluded from the data distribution. The resultant images are composed entirely of non-target-class objects, (e.g., cylinders and spheres).

Fig. B-7 presents high-confidence misclassifications for each classes of MNIST, with digit 0-4 on the top two rows and digit 5-9 on the bottom two rows.

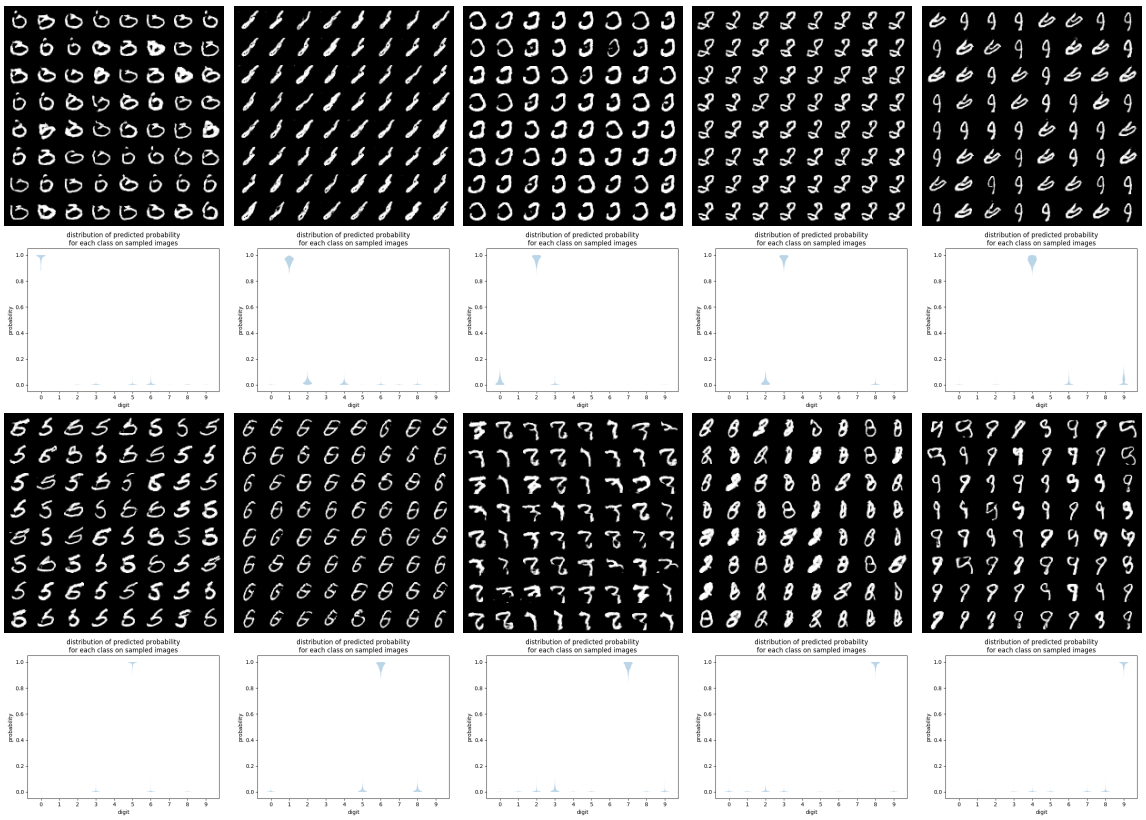


Figure B-7: Examples and violin plots for high confidence misclassified examples. Top two rows: 0-4; bottom two rows: 5-9.

Fig. B-8 presents high-confidence misclassifications for each classes of Fashion-MNIST, with T-shirt, trousers pullover, dress and coat on the top two rows and sandal, shirt, sneaker, bag and ankle boot on the bottom two rows. The confidence plot for the trousers samples indicates that the sampling is not successful.

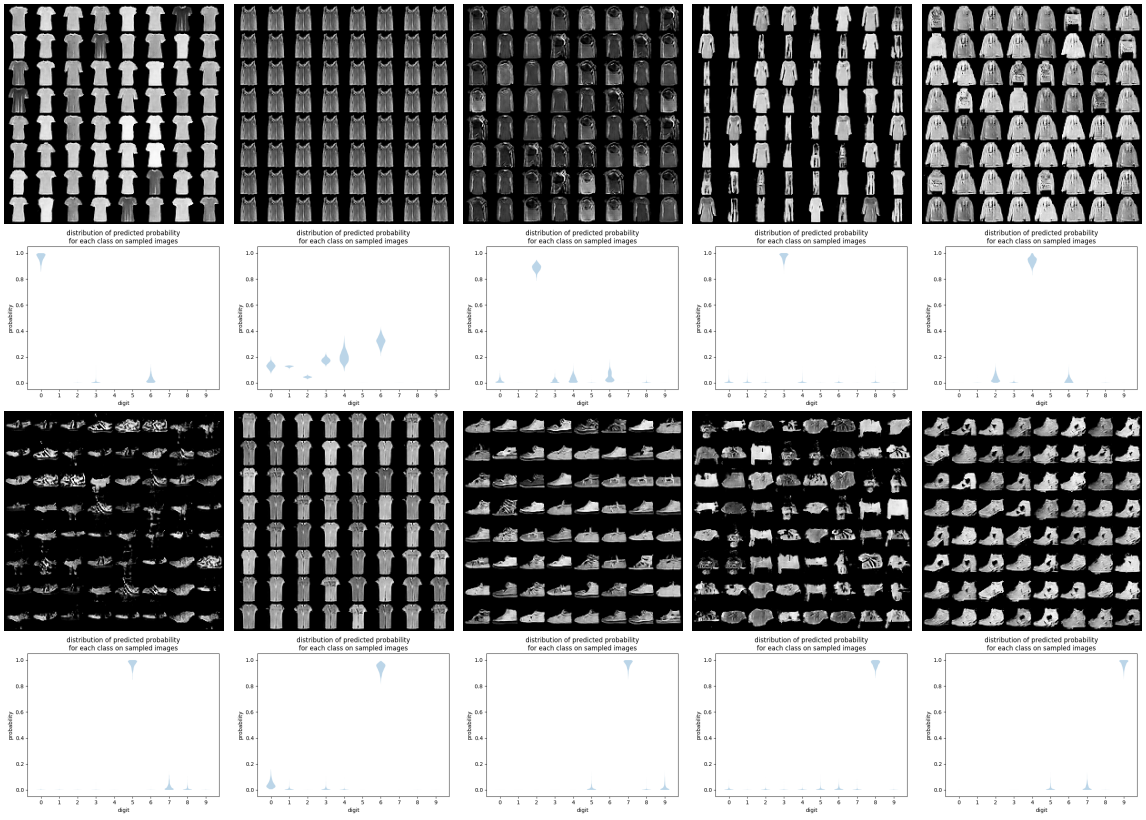
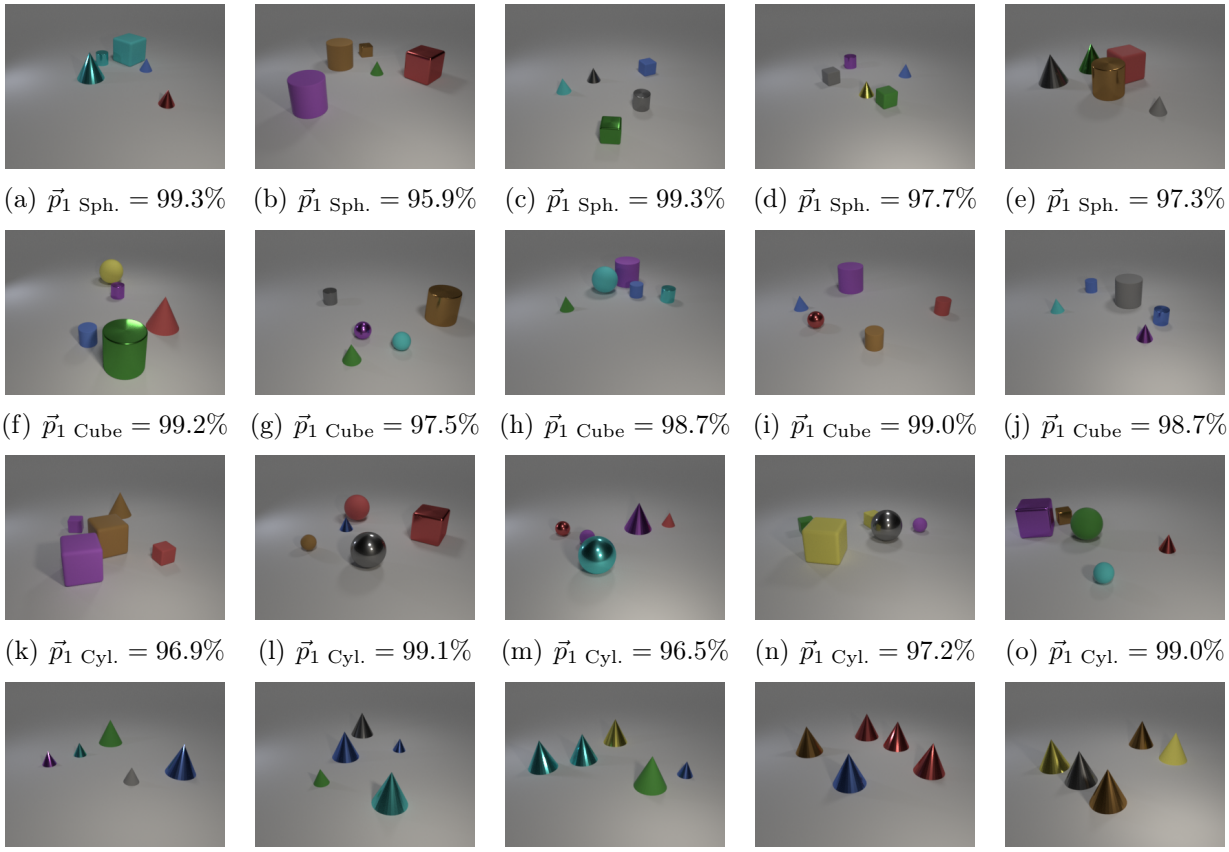


Figure B-8: Samples and violin plots for high confidence misclassified examples. Top row: T-shirt, trousers (sample failure), pullover, dress, coat. Bottom row: sandal, shirt, sneaker, bag, ankle boot.

B.7 Novel Class Extrapolation Analysis

Figure B-9 shows novel class extrapolation examples for CLEVR.



(a) $\vec{p}_{1\ \text{Sph.}} = 99.3\%$ (b) $\vec{p}_{1\ \text{Sph.}} = 95.9\%$ (c) $\vec{p}_{1\ \text{Sph.}} = 99.3\%$ (d) $\vec{p}_{1\ \text{Sph.}} = 97.7\%$ (e) $\vec{p}_{1\ \text{Sph.}} = 97.3\%$

(f) $\vec{p}_{1\ \text{Cube}} = 99.2\%$ (g) $\vec{p}_{1\ \text{Cube}} = 97.5\%$ (h) $\vec{p}_{1\ \text{Cube}} = 98.7\%$ (i) $\vec{p}_{1\ \text{Cube}} = 99.0\%$ (j) $\vec{p}_{1\ \text{Cube}} = 98.7\%$

(k) $\vec{p}_{1\ \text{Cyl.}} = 96.9\%$ (l) $\vec{p}_{1\ \text{Cyl.}} = 99.1\%$ (m) $\vec{p}_{1\ \text{Cyl.}} = 96.5\%$ (n) $\vec{p}_{1\ \text{Cyl.}} = 97.2\%$ (o) $\vec{p}_{1\ \text{Cyl.}} = 99.0\%$

(p) $\vec{p}_{5\ \text{Cubes}} = 74.6\%$ (q) $\vec{p}_{5\ \text{Cubes}} = 89.5\%$ (r) $\vec{p}_{5\ \text{Cubes}} = 93.3\%$ (s) $\vec{p}_{5\ \text{Cubes}} = 91.6\%$ (t) $\vec{p}_{5\ \text{Cubes}} = 89.9\%$

Figure B-9: Sampled novel class extrapolation examples and their associated prediction confidences. Similar to high confidence misclassified examples, for each target constraint (e.g., "1 Cube"), we remove examples of the target class (e.g., cubes) from the data distribution, but add to the cone object to it, a novel class not present in the training distribution. B-9(n) is the only example which by chance does not include a novel class object.

Fig. B-10 shows examples for novel-class extrapolation on MNIST. The classifier is trained on digit 0, 1, 3, 6 and 9, and tested on images generated by a GAN trained on digit 2, 4, 5, 7 and 8.

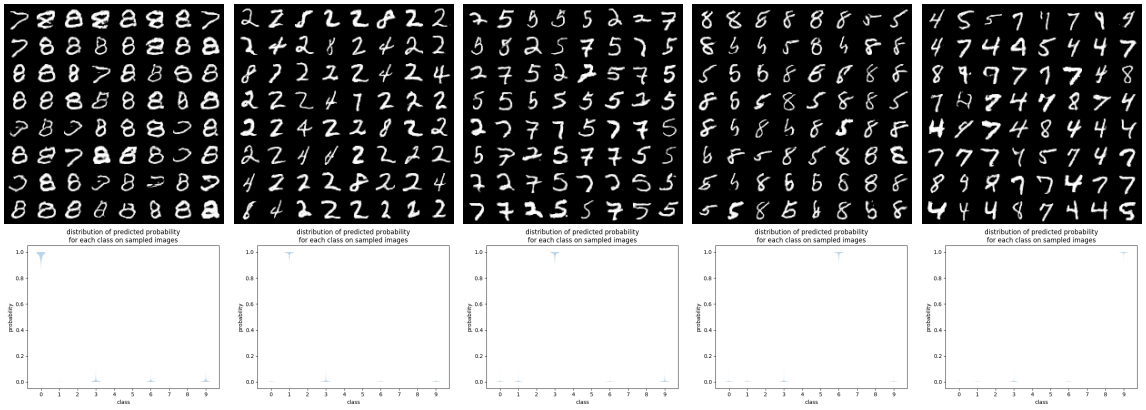


Figure B-10: Samples and confidence plots for MNIST novel class extrapolation for digits 0, 1, 3, 6 and 9, in that order.

Fig. B-11 shows examples for novel-class extrapolation on Fashion-MNIST. The classifier is trained on pullover, dress, sandal, shirt and ankle boot, and tested on images generated by a GAN trained on T-shirt, trousers, coat, sneaker and bag.

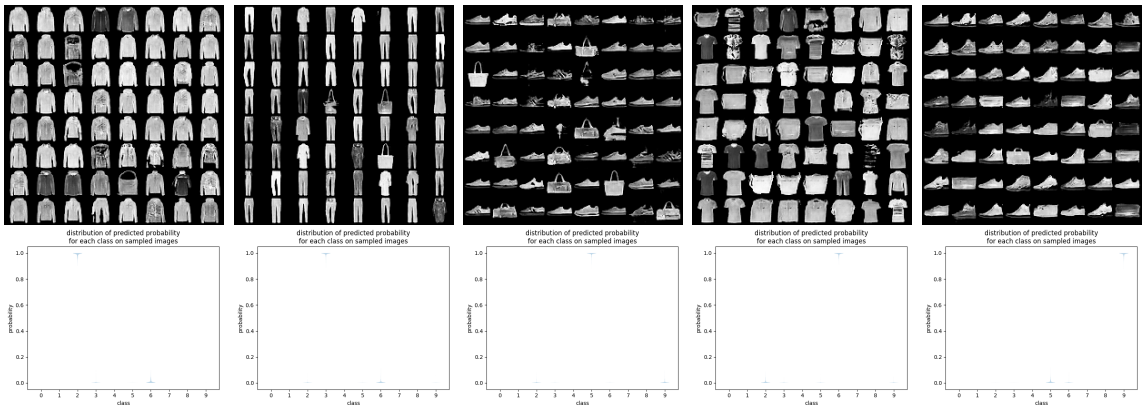


Figure B-11: Samples and confidence plots for Fashion-MNIST novel class extrapolation for pullover, dress, sandal, shirt and ankle boot, in that order.

B.8 Domain Adaptation Analysis

Fig. B-12 and B-13 show additional samples and confidence plots for the baseline and ADDA model, respectively. Top two rows are for digit 0-4, and bottom two rows are for digit 5-9.

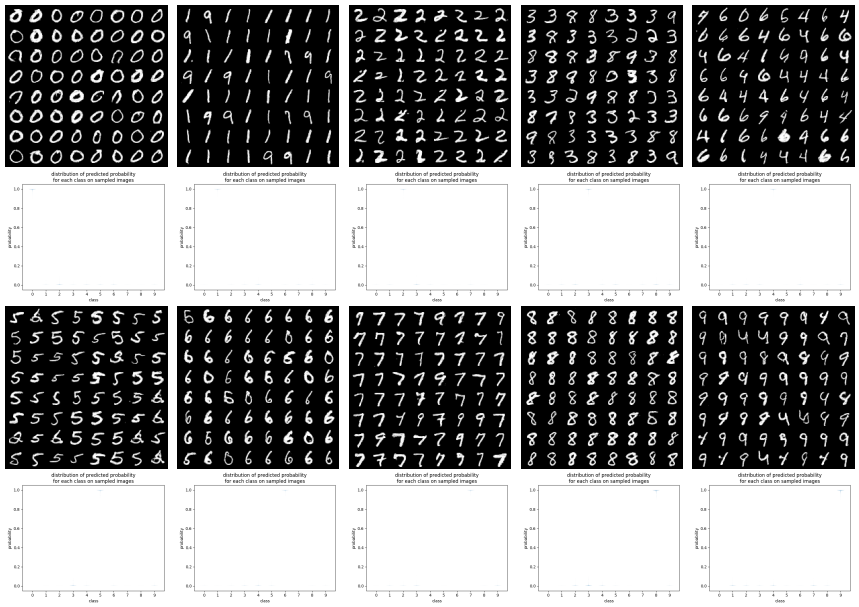


Figure B-12: High confident MNIST samples generated for each class as predicted by the baseline model.

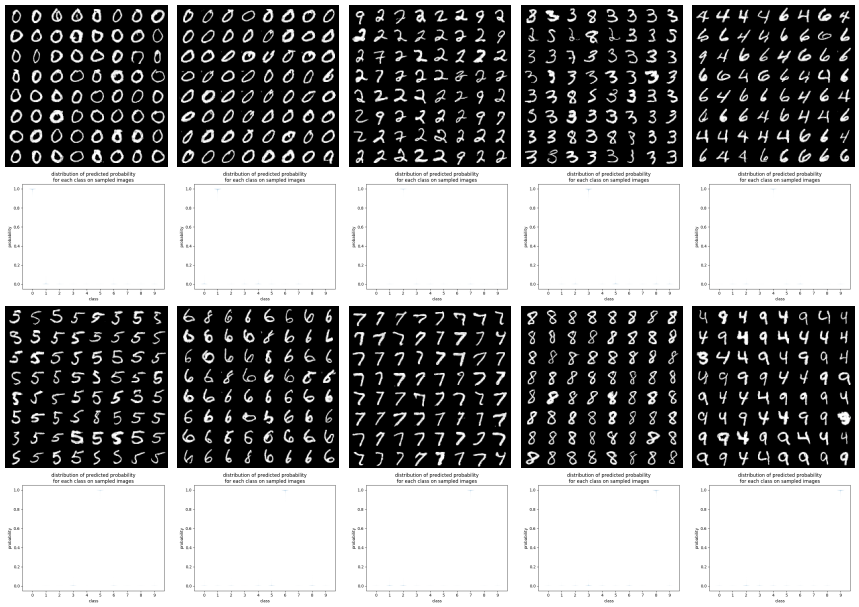


Figure B-13: High confident MNIST samples generated for each class as predicted by the ADDA model.

B.9 Quantitative Prediction Confidence Summary

Tables B.4, B.5, and B.6 present the extension of Table 4.2 in Sec. 4.4.9. These results show that the inferred samples have predicted confidence closely matching the specified confidence targets. This indicates the MCMC methods used by BAYES-TREX are successful for the tested domains and scenarios. Queries for 5 Cubes in the novel class extrapolation CLEVR experiments use a stopping criterion of 1500 samples instead of the standard 500 (Fig. B-14). Averages reported across 10 inference runs. Fig. B-15 presents the prediction confidence for pairwise ambiguous samples for MNIST and Fashion-MNIST.

Table B.4: Prediction confidence for samples on high-confidence examples (left) and high confidence misclassifications (right).

Target	Prediction Confidence
$\vec{p}_0 = 1$	0.999 ± 0.006
$\vec{p}_1 = 1$	0.999 ± 0.003
$\vec{p}_2 = 1$	0.999 ± 0.006
$\vec{p}_3 = 1$	0.999 ± 0.005
$\vec{p}_4 = 1$	0.998 ± 0.008
$\vec{p}_5 = 1$	0.999 ± 0.006
$\vec{p}_6 = 1$	0.998 ± 0.007
$\vec{p}_7 = 1$	0.998 ± 0.007
$\vec{p}_8 = 1$	0.999 ± 0.004
$\vec{p}_9 = 1$	0.998 ± 0.007
$\vec{p}_{\text{T-Shirt}} = 1$	0.991 ± 0.016
$\vec{p}_{\text{Trouser}} = 1$	0.999 ± 0.006
$\vec{p}_{\text{Pullover}} = 1$	0.984 ± 0.019
$\vec{p}_{\text{Dress}} = 1$	0.993 ± 0.008
$\vec{p}_{\text{Coat}} = 1$	0.983 ± 0.021
$\vec{p}_{\text{Sandal}} = 1$	0.998 ± 0.008
$\vec{p}_{\text{Shirt}} = 1$	0.987 ± 0.020
$\vec{p}_{\text{Sneaker}} = 1$	0.994 ± 0.016
$\vec{p}_{\text{Bag}} = 1$	0.999 ± 0.006
$\vec{p}_{\text{Ankle Boot}} = 1$	0.996 ± 0.012
$\vec{p}_{5\text{ Spheres}} = 1$	0.943 ± 0.020
$\vec{p}_{2\text{ Blue Spheres}} = 1$	0.892 ± 0.245

Target	Prediction Confidence
$\vec{p}_0 = 1$	0.981 ± 0.027
$\vec{p}_1 = 1$	0.953 ± 0.028
$\vec{p}_2 = 1$	0.968 ± 0.028
$\vec{p}_3 = 1$	0.969 ± 0.027
$\vec{p}_4 = 1$	0.955 ± 0.030
$\vec{p}_5 = 1$	0.990 ± 0.018
$\vec{p}_6 = 1$	0.970 ± 0.026
$\vec{p}_7 = 1$	0.968 ± 0.029
$\vec{p}_8 = 1$	0.982 ± 0.024
$\vec{p}_9 = 1$	0.983 ± 0.022
$\vec{p}_{\text{T-Shirt}} = 1$	0.964 ± 0.029
$\vec{p}_{\text{Trouser}} = 1$	(sample failure)
$\vec{p}_{\text{Pullover}} = 1$	0.886 ± 0.027
$\vec{p}_{\text{Dress}} = 1$	0.970 ± 0.026
$\vec{p}_{\text{Coat}} = 1$	0.938 ± 0.030
$\vec{p}_{\text{Sandal}} = 1$	0.968 ± 0.030
$\vec{p}_{\text{Shirt}} = 1$	0.938 ± 0.032
$\vec{p}_{\text{Sneaker}} = 1$	0.969 ± 0.028
$\vec{p}_{\text{Bag}} = 1$	0.967 ± 0.026
$\vec{p}_{\text{Ankle Boot}} = 1$	0.971 ± 0.027
$\vec{p}_{1\text{ Cube}} = 1$	0.929 ± 0.062
$\vec{p}_{1\text{ Cylinder}} = 1$	0.972 ± 0.021
$\vec{p}_{1\text{ Sphere}} = 1$	0.843 ± 0.266
$\vec{p}_{2\text{ Cylinders}} = 1$	0.545 ± 0.230

Table B.5: (Fashion-)MNIST confidence interpolation.

Target	Prediction Confidence
$\vec{p}_8 = 0.0, \vec{p}_9 = 1.0$	$(0.002 \pm 0.006, 0.990 \pm 0.016)$
$\vec{p}_8 = 0.1, \vec{p}_9 = 0.9$	$(0.030 \pm 0.039, 0.936 \pm 0.051)$
$\vec{p}_8 = 0.2, \vec{p}_9 = 0.8$	$(0.170 \pm 0.039, 0.788 \pm 0.040)$
$\vec{p}_8 = 0.3, \vec{p}_9 = 0.7$	$(0.275 \pm 0.041, 0.682 \pm 0.040)$
$\vec{p}_8 = 0.4, \vec{p}_9 = 0.6$	$(0.378 \pm 0.040, 0.578 \pm 0.040)$
$\vec{p}_8 = 0.5, \vec{p}_9 = 0.5$	$(0.477 \pm 0.039, 0.477 \pm 0.039)$
$\vec{p}_8 = 0.6, \vec{p}_9 = 0.4$	$(0.581 \pm 0.038, 0.374 \pm 0.039)$
$\vec{p}_8 = 0.7, \vec{p}_9 = 0.3$	$(0.680 \pm 0.041, 0.275 \pm 0.039)$
$\vec{p}_8 = 0.8, \vec{p}_9 = 0.2$	$(0.788 \pm 0.040, 0.167 \pm 0.041)$
$\vec{p}_8 = 0.9, \vec{p}_9 = 0.1$	$(0.926 \pm 0.050, 0.039 \pm 0.040)$
$\vec{p}_8 = 1.0, \vec{p}_9 = 0.0$	$(0.989 \pm 0.016, 0.002 \pm 0.007)$

Target	Prediction Confidence
$\vec{p}_{\text{T-Shirt}} = 0.0, \vec{p}_{\text{Trousers}} = 1.0$	$(0.001 \pm 0.004, 0.995 \pm 0.012)$
$\vec{p}_{\text{T-Shirt}} = 0.1, \vec{p}_{\text{Trousers}} = 0.9$	$(0.026 \pm 0.035, 0.950 \pm 0.050)$
$\vec{p}_{\text{T-Shirt}} = 0.2, \vec{p}_{\text{Trousers}} = 0.8$	$(0.166 \pm 0.040, 0.791 \pm 0.041)$
$\vec{p}_{\text{T-Shirt}} = 0.3, \vec{p}_{\text{Trousers}} = 0.7$	$(0.275 \pm 0.037, 0.686 \pm 0.038)$
$\vec{p}_{\text{T-Shirt}} = 0.4, \vec{p}_{\text{Trousers}} = 0.6$	$(0.379 \pm 0.038, 0.586 \pm 0.038)$
$\vec{p}_{\text{T-Shirt}} = 0.5, \vec{p}_{\text{Trousers}} = 0.5$	$(0.436 \pm 0.040, 0.459 \pm 0.040)$
$\vec{p}_{\text{T-Shirt}} = 0.6, \vec{p}_{\text{Trousers}} = 0.4$	$(0.583 \pm 0.038, 0.382 \pm 0.037)$
$\vec{p}_{\text{T-Shirt}} = 0.7, \vec{p}_{\text{Trousers}} = 0.3$	$(0.685 \pm 0.039, 0.281 \pm 0.040)$
$\vec{p}_{\text{T-Shirt}} = 0.8, \vec{p}_{\text{Trousers}} = 0.2$	$(0.790 \pm 0.037, 0.177 \pm 0.037)$
$\vec{p}_{\text{T-Shirt}} = 0.9, \vec{p}_{\text{Trousers}} = 0.1$	$(0.936 \pm 0.045, 0.029 \pm 0.041)$
$\vec{p}_{\text{T-Shirt}} = 1.0, \vec{p}_{\text{Trousers}} = 0.0$	$(0.985 \pm 0.019, 0.000 \pm 0.003)$

Table B.6: Prediction confidence for novel class extrapolation (left) and domain adaptation (right).

Target	Prediction Confidence
$\vec{p}_0 = 1$	0.976 ± 0.025
$\vec{p}_1 = 1$	0.988 ± 0.186
$\vec{p}_3 = 1$	0.987 ± 0.020
$\vec{p}_6 = 1$	0.989 ± 0.018
$\vec{p}_9 = 1$	0.995 ± 0.013
$\vec{p}_{\text{Pullover}} = 1$	0.991 ± 0.016
$\vec{p}_{\text{Dress}} = 1$	0.994 ± 0.013
$\vec{p}_{\text{Sandal}} = 1$	0.995 ± 0.013
$\vec{p}_{\text{Shirt}} = 1$	0.994 ± 0.012
$\vec{p}_{\text{Ankle Boot}} = 1$	0.993 ± 0.015
$\vec{p}_{1\text{ Cube}} = 1$	0.983 ± 0.014
$\vec{p}_{1\text{ Cylinder}} = 1$	0.959 ± 0.031
$\vec{p}_{1\text{ Sphere}} = 1$	0.969 ± 0.022
$\vec{p}_{5\text{ Cubes}} = 1$	0.921 ± 0.029

Target	Prediction Confidence
$\vec{p}_0 = 1$	0.996 ± 0.011
$\vec{p}_1 = 1$	0.994 ± 0.014
$\vec{p}_2 = 1$	0.998 ± 0.008
$\vec{p}_3 = 1$	0.994 ± 0.015
$\vec{p}_4 = 1$	0.997 ± 0.010
$\vec{p}_5 = 1$	0.998 ± 0.007
$\vec{p}_6 = 1$	0.996 ± 0.011
$\vec{p}_7 = 1$	0.996 ± 0.011
$\vec{p}_8 = 1$	0.995 ± 0.013
$\vec{p}_9 = 1$	0.996 ± 0.012

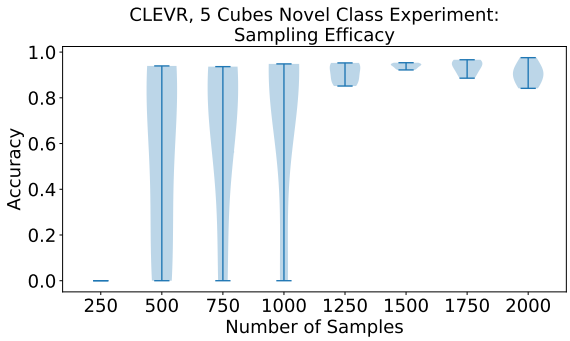


Figure B-14: We assess sample efficacy which enables us to approximate the prediction target while minimizing compute utilization. For this experiment, the target is a novel class extrapolation CLEVR scene classified as containing 5 cubes with high confidence ($\vec{p}_{5\text{ Cubes}} = 1$), but which consists of only spheres, cylinders, and cones. For each evaluated number of samples, we conduct 5 independent Bayes-TrEx runs. This evaluation shows that 1250 or more samples are needed for this trial.

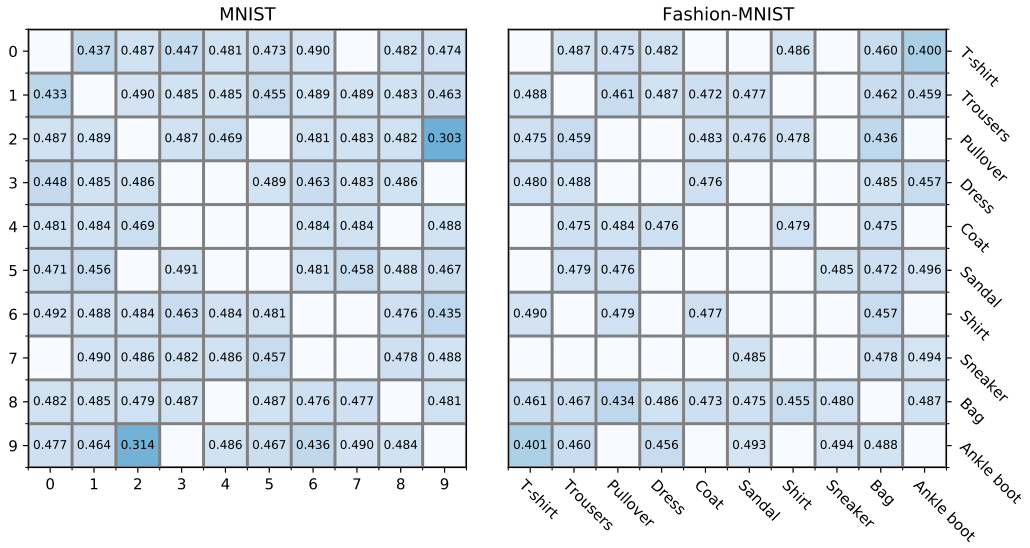


Figure B-15: Prediction confidence for (Fashion-)MNIST ambiguous samples. For each class combination, the lower left triangle shows the the confidence for the digit denoted on the horizontal axis, and the upper right triangle shows the confidence for the digit on the vertical axis. For example, for the MNIST class combination 9v0, the classifier confidence in class 0 is 0.477 (bottom left) while the classifier confidence in class 9 is 0.474 (top right). Diagonal entries are blank since they have the same class on row and column. Off-diagonal blank entries indicate that Bayes-TrEx sampling failed for that class pair.

B.10 Test Set Evaluation

Tab. B.7 extends Tab. 4.5 in Sec. 4.4.10 and includes misclassified vs. mislabeled images of all (Fashion-)MNIST classes.

Table B.7: An alternative to using BAYES-TREX for finding highly confident classification failures is to evaluate the high confidence example confusion matrix and associated images from the test set. Here, we show all 'misclassified' examples where the classifier failed to predict the given label for the MNIST and Fashion-MNIST datasets. For MNIST, we observe that the majority (60/84) of these images are mislabeled: for example, all of the labeled 2s clearly belong to other classes (8, 7, 7, 3, 1, 7, 7, 7, respectively). While MNIST had 84 total misclassifications, Fashion-MNIST had 802 total misclassifications. We randomly select 10 misclassifications from each class for analysis (with the exception of the "trousers" class, as there were 3 total misclassifications for this label). While Fashion-MNIST is more balanced, we again observe a majority of examples to be mislabeled ground truth (52/93) instead of misclassifications.

Class	Misclassified	Mislabeled
0		
1		
2	∅	
3		
4		
5		
6		
7		
8	∅	
9		
Tshirt		
Trouser	∅	
Pullover		
Dress		
Coat		
Sandal		
Shirt		
Sneaker		
Bag		
Boot		∅

Figure B-16 shows the confusion matrix of the MNIST (left) and Fashion-MNIST (right) classifiers.

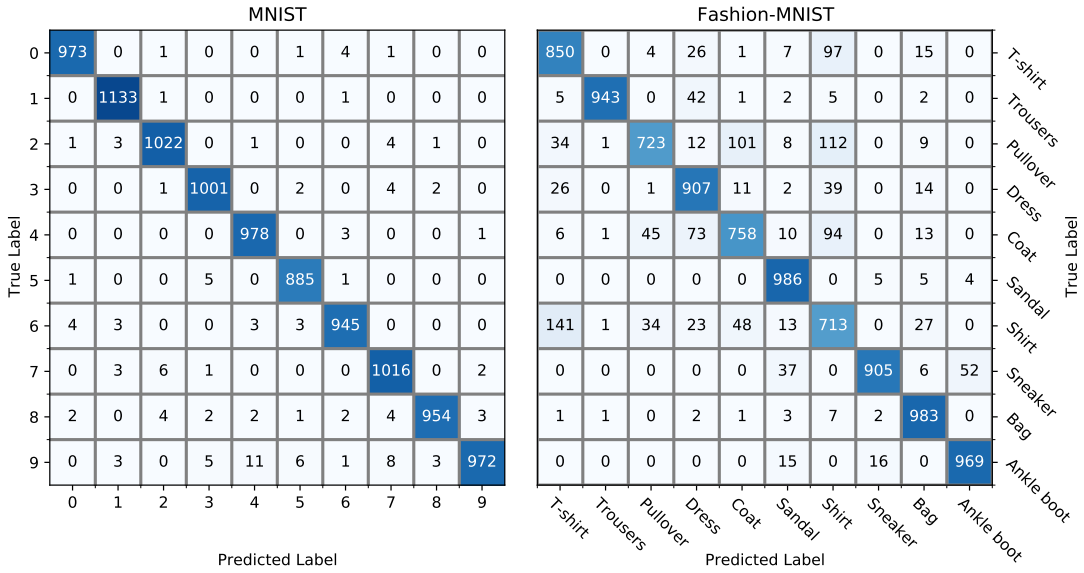


Figure B-16: Confusion matrices for MNIST (left) and Fashion-MNIST (right) classifiers. Note that these matrices include all test set examples, not just those which evoke high confidence responses from the classifier.

B.11 Bayes-TrEx with Saliency Maps

We demonstrate a simple use case of combining with Bayes-TrEx samples with downstream interpretability methods. Fig. B-17 (left) shows an image for which the classifier mistakes it to contain one cube with 93.5% accuracy. Fig. B-17 (middle) presents its SmoothGrad [186] saliency map and Fig. B-17 (right) overlays it on top of the image. We can see that the most salient part contributing to the 1-cube decision is the front red cylinder. Indeed, as we confirm in Fig. B-18, among all single object removals, removing this object has the biggest effect to the classifier confidence, decreasing it to 29.0%.

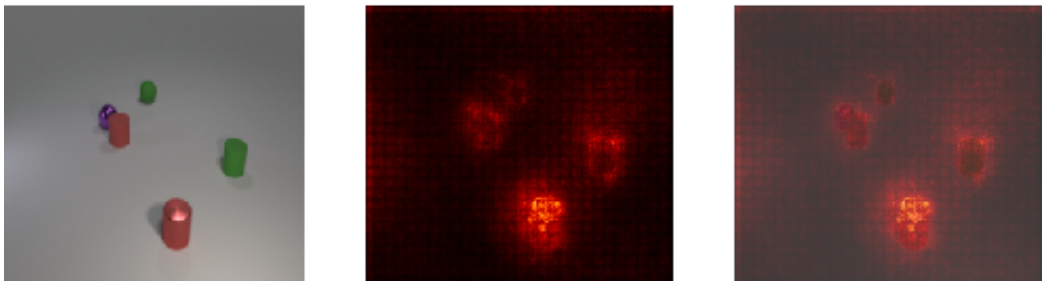
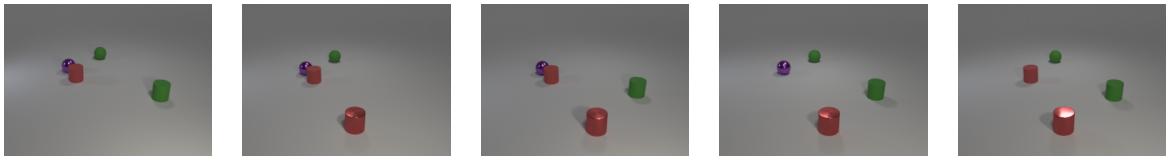


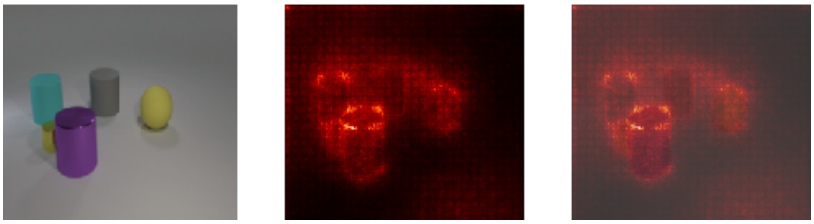
Figure B-17: Left: the original image, preprocessed for classification by resizing and normalizing. The classifier is 93.5% confident this scene contains 1 cube, when in fact it is composed of 3 cylinders and 2 spheres. Middle: the SmoothGrad saliency map for this input. Right: the saliency map overlaid upon the original image. This saliency map most strongly highlights the red metal cylinder, indicating that this cylinder is likely the cause of the misclassification.



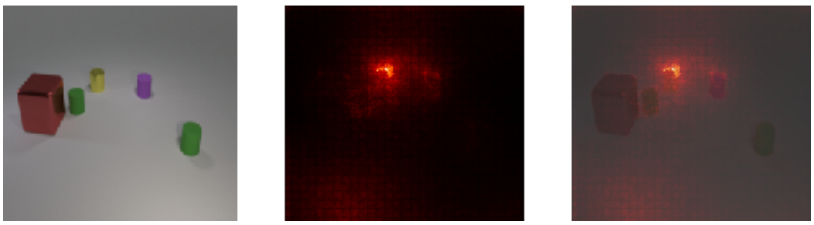
(a) $\vec{p}_{\text{1 Cube}} = 29.0\%$ (b) $\vec{p}_{\text{1 Cube}} = 68.5\%$ (c) $\vec{p}_{\text{1 Cube}} = 81.2\%$ (d) $\vec{p}_{\text{1 Cube}} = 99.0\%$ (e) $\vec{p}_{\text{1 Cube}} = 99.4\%$

Figure B-18: Prediction confidence for 1-cube after every single object is removed in turn. As suggested by the saliency map, the removal of the red metal cylinder most prominently reduces the classification confidence, from 93.5% to 29.0%.

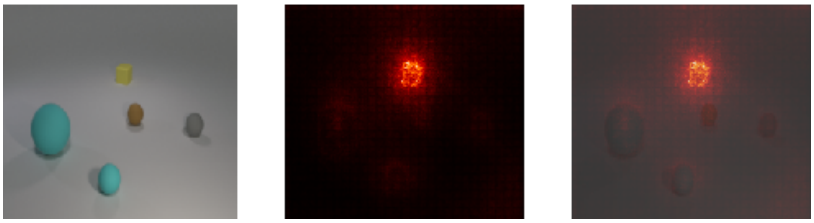
Fig. B-19 presents additional case studies with the same setup. Note that Fig. B-19(e) shows a failure of SmoothGrad.



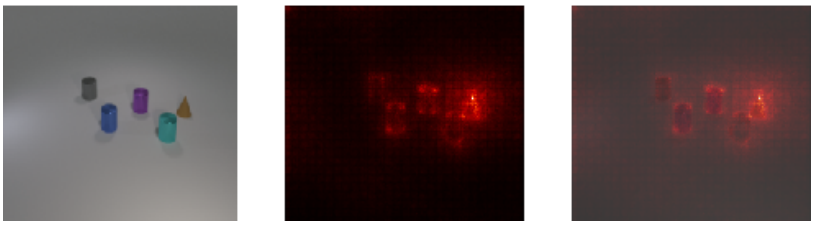
(a) Original image: $\vec{p}_{1\ \text{Cube}} = 85.5\%$. Purple cylinder removed: $\vec{p}_{1\ \text{Cube}} = 1.9\%$



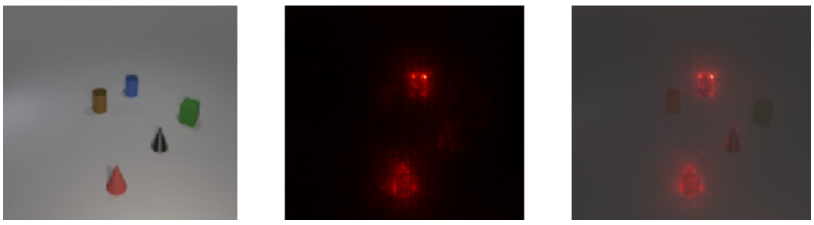
(b) Original image: $\vec{p}_{1\ \text{Sphere}} = 97.9\%$. Yellow cylinder removed: $\vec{p}_{1\ \text{Sphere}} = 5.2\%$



(c) Original image: $\vec{p}_{1\ \text{Cylinder}} = 85.4\%$. Red sphere removed: $\vec{p}_{1\ \text{Cylinder}} = 0.9\%$



(d) Original image: $\vec{p}_{1\ \text{Cube}} = 99.7\%$. Cone removed: $\vec{p}_{1\ \text{Cube}} = 0.4\%$



(e) Original image: $\vec{p}_{1\ \text{Sphere}} = 98.0\%$. Gray cone removed: $\vec{p}_{1\ \text{Sphere}} = 0.3\%$

Figure B-19: Images sampled with BAYES-TREX and their saliency maps. B-19(a)-B-19(c) are high confidence misclassified examples; B-19(d)-B-19(e) are novel class extrapolation examples. In B-19(e), the saliency map primarily highlights two objects: the red cone and the blue cylinder. Removing either of these objects does not result in a change of prediction. Instead, the misclassification of 1 Sphere is due to the marginally-highlighted gray cone.

B.12 Ethics Statement

Bayes-TrEx has potential to allow humans to build more accurate mental models of how neural networks make decisions. Further, Bayes-TrEx can be useful for debugging, interpreting, and understanding networks—all of which can help us build *better*, less biased, increasingly human-aligned models.

However, Bayes-TrEx is subject to the same caveats as typical software testing approaches: the absence of exposed bad samples does not mean the system is free from defects. One concern is how system designers and users will interact with Bayes-TrEx in practice. If Bayes-TrEx does not reveal degenerate examples, these stakeholders might develop inordinate trust [120] in their models.

Additionally, one Bayes-TrEx use case is to generate examples to be used as inputs to downstream local explanation methods. As a community, we know that many of these local explanations can be challenging to understand [157, 154], misleading [1, 100, 175], or susceptible to adversarial attacks [185]. In human-human interaction, even nonsensical explanations can increase compliance [117]. As we build post-hoc explanation techniques, we must evaluate whether the produced explanations help humans moderate trust and act appropriately—for example, by overriding the model's decisions.

Appendix C

Inspect: Robot Controllers

C.1 Scale-Invariance and the Volume Interpretation of α

We show that Eq. 5.3 results in the formulation being scale-invariant with respect to b. Consider the same behavior under two different units b_1 and b_2 with $b_1 = c \cdot b_2$. For example, b_1 can be the trajectory length in centimeters and b_2 is the same quantity but in meters, and $c = 100$. Thus, $p(c \cdot b_1) = p(b_2)$ and $b_1^* = c \cdot b_2^*$. To maintain the same α level in Eq. 5.3, we need to have $\sigma_1 = c \cdot \sigma_2$. This implies that

$$p(t, \tau | \hat{b}_1 = b_1^*) = \frac{\mathcal{N}(b_1^*; b(\tau, t), \sigma_1^2) p(\tau|t)\pi(t)}{p(\hat{b}_1 = b_1^*)} \tag{C.1}$$

$$= \frac{\mathcal{N}(b_2^*; b(\tau, t), \sigma_2^2) p(\tau|e)\pi(t)}{p(\hat{b}_2 = b_2^*)} = p(t|\hat{b}_2 = b_2^*) \tag{C.2}$$

because $\mathcal{N}(b_1^*; b(\tau, t), \sigma_1^2) = \mathcal{N}(b_2^*; b(\tau, t), \sigma_2^2)$ due to the same scaling of $b_1 \sim b_2$ and $\sigma_1 \sim \sigma_2$, and $p(\hat{b}_1 = b_1^*) = p(\hat{b}_2 = b_2^*)$ as they are the same event. We conclude that the posterior distribution is scale-invariant with respect to $b(\tau, t)$.

To motivate the bound of $[b^* - \sqrt{3}\sigma, b^* + \sqrt{3}\sigma]$ in Eq. 5.3, we consider a uniform approximation to $\mathcal{N}(b^*, \sigma^2)$. To match the mean b^* and standard deviation σ, $\mathcal{U}(b^* - \sqrt{3}\sigma, b^* + \sqrt{3}\sigma)$ is needed. If we use this uniform distribution in Eq. 5.2 in lieu of the normal distribution, the posterior can be instantiated by sampling from the prior and rejecting tasks for which the trajectory behavior $b(\tau, t)$ falls outside of this bound. Thus, Eq. 5.3 specifies that the "volume" of $(\alpha \cdot 100)\%$ under $p(t, \tau)$ is maintained.

The same invariance and "volume" interpretation holds for Eq. 5.5 as well. The former stems from the standardization on b performed in Eq. 5.4. The latter uses the same uniform approximation but the bound is one-sided since $\beta \in (0, 1)$ by nature of the sigmoid transformation.

C.2 MCMC Sampling with Stochastic Dynamics

Using the same logic as the case of stochastic controller, RoCUS can also accommodate stochasticity in transition dynamics (e.g. object position uncertainty after it is pushed), *as long as such stochasticity can be captured in a random variable v and $p(v|t)$ can be evaluated.* This is typically possible in simulation, and the modification to Alg. 1 is similar to the case of stochastic controllers. In the real world, we can

- treat a sampled trajectory as the deterministic one;
- restart multiple times to estimate $\mathbb{E}_\tau[b(\tau, t)]$; or
- use likelihood-free MCMC methods [28].

We leave these investigations to future work, and use deterministic dynamics in our experiments.

C.3 Mathematical Definitions of Behaviors

A versatile and general form of a behavior is the (normalized or unnormalized) line integral of some scalar field along the trajectory. Specifically, we have

$$b = \int_\tau V(\vec{x})ds \quad \text{or} \quad b = \frac{1}{||\tau||}\int_\tau V(\vec{x})ds. \tag{C.3}$$

Using this general definition, we define a list of behaviors in Tab. C.1.

Name	Definition	Name	Definition
Trajectory Length	$b = \int_\tau 1 ds$	Straight-Line Deviation	$b = \frac{1}{\|\|\tau\|\|}\int_\tau \|\|\vec{x} - \text{proj}_{\vec{x}_f - \vec{x}_i}\vec{x}\|\| ds$
Average Velocity	$b = \frac{1}{\|\|\tau\|\|}\int_\tau \|\|\dot{\vec{x}}\|\| ds$	Obstacle Clearance	$b = \frac{1}{\|\|\tau\|\|}\int_\tau \min_{\vec{x}_o \in \mathcal{O}} \|\|\vec{x} - \vec{x}_o\|\| ds$
Average Acceleration	$b = \frac{1}{\|\|\tau\|\|}\int_\tau \|\|\ddot{\vec{x}}\|\| ds$	Near-Obstacle Velocity	$b = \frac{\int_\tau \|\|\dot{\vec{x}}\|\| / \min_{\vec{x}_o \in \mathcal{O}} \|\|\vec{x} - \vec{x}_o\|\| ds}{\int_\tau 1 / \min_{\vec{x}_o \in \mathcal{O}} \|\|\vec{x} - \vec{x}_o\|\| ds}$
Average Jerk	$b = \frac{1}{\|\|\tau\|\|}\int_\tau \|\| \dddot{\vec{x}} \|\| ds$	Motion Legibility	$b = \frac{1}{\|\|\tau\|\|}\int_\tau p(g\|\vec{x}) ds$

Table C.1: A list of behavior definitions.

Trajectory length simply measures how long the trajectory is. In most of the behaviors below, the normalizing factor is also length to decorrelate the behavior value from it.

Average velocity, acceleration and jerk are useful for a general understanding about how fast and abruptly the robot moves, which is an important factor to its safety.

Straight-line deviation measures how much the robot trajectory deviates from the straight-line path, in either the task space or the state space. A specific task instance in which the straight-line path is feasible (e.g. with no obstacles) is typically

considered easy. Thus, we can find tasks of varying difficulty level on the spectrum of deviation values. In the definition, $\vec{x}_i$ is the initial state, $\vec{x}_f$ is the final state, and proj is the projection operator.

Obstacle clearance measures the average distance to the closest obstacle. Finding situations in which the robot moves very close to obstacles is crucial to understanding the collision risk level. In the definition, $\mathcal{O}$ represents the obstacle space.

Near-obstacle velocity calculates how fast the robot moves around obstacles. We define it as the average velocity on the trajectory weighted by the inverse distance to the closest obstacle. Other weighting method can be used, as long as it is non-negative and monotonically decreasing with distance. This behavior is correlated with the damage of a potential collision, as high-speed collisions are usually far more dangerous and costly. Since we want the value to represent the average velocity, we normalize by the integral of weights along the trajectory.

Motion legibility measures how well the goal can be predicted over the course of the exhibited trajectory. In our definition, we use $p(g|\vec{x})$, or the conditional probability of the goal g given at the current robot state $\vec{x}$, but there may be better application-specific definitions.

C.4 Dynamical System Modulation

We review the DS formulation proposed by [86], and present our problem-specific adaptations for 2D Navigation in App. C.8.2 and 7DoF arm reaching in App. C.10.3. A reader familiar with DS motion controllers may skip this review.

Given a target $\vec{x}^*$ and the robot's current state $\vec{x}$, a linear controller $\vec{u}(x) = \vec{x}^* - \vec{x}$ will guarantee convergence of $\vec{x}$ to $\vec{x}^*$ if there are no obstacles. However, it can easily get stuck in the presence of obstacles. [86] proposes a method to calculate a modulation matrix $M(\vec{x})$ at every $\vec{x}$ such that if the new controller follows $\vec{u}_M(\vec{x}) = M(\vec{x}) \cdot \vec{u}(\vec{x})$, then $\vec{x}$ still converges to $\vec{x}^*$ but never gets stuck, as long as $\vec{x}^*$ is in free space. In short, the objective of the DS modulation is to preserve the linear controller's convergence guarantee while also ensuring that the robot is never in collision.

The modulation matrix $M(\vec{x})$ is computed from a list of obstacles, each of which is represented by a Γ-function. For the i-th obstacle $\mathcal{O}_i$, its associated gamma function Γ_i must satisfy the following properties:

- $\Gamma_i(\vec{x}) \leq 1 \iff \vec{x} \in \mathcal{O}_i$,
- $\Gamma_i(\vec{x}) = 1 \iff \vec{x} \in \partial\mathcal{O}_i$,
- $\exists\, \vec{r}_i, \text{s.t.} \forall\, t_1 \geq t_2 \geq 0, \forall\, \vec{u}, \Gamma_i(\vec{r}_i + t_1\vec{u}) \geq \Gamma_i(\vec{r}_i + t_2\vec{u})$.

In words, the Γ-function value needs to be less than 1 when inside the obstacle, equal to 1 on the boundary, greater than 1 when outside. This function must also be monotonically increasing radially outward from a specific point $\vec{r}_i$. This point is dubbed the *reference point*. From this formulation, $\vec{r}_i \in \mathcal{O}_i$ and any ray from $\vec{r}_i$ intersects with the obstacle boundary $\partial\mathcal{O}_i$ exactly once. The latter property is also the definition that $\mathcal{O}_i$ is "star-shaped" (Fig. C-4). For most common (2D) geometric

shape such as rectangles, circles, ellipses, regular polygons and regular stars, $\vec{r}_i$ can be chosen as the geometric center.

We first consider the case of a single obstacle $\mathcal{O}$, represented by Γ with reference point $\vec{r}$. Use d to denote the dimension of the space. We define

$$M(\vec{x}) = E(\vec{x})D(\vec{x})E^{-1}(\vec{x}). \tag{C.4}$$

We have

$$E(x) = [\vec{s}(\vec{x}), \vec{e}_1(\vec{x}), ..., \vec{e}_{d-1}(\vec{x})], \tag{C.5}$$

where

$$\vec{s}(\vec{x}) = \frac{\vec{x} - \vec{r}}{||\vec{x} - \vec{r}||} \tag{C.6}$$

is the unit vector in the direction of $\vec{x}$ from $\vec{r}$, and $\vec{e}_1(\vec{x}), ..., \vec{e}_{d-1}(\vec{x})$ form a $d-1$ orthonormal basis to the gradient of the Γ-function, $\nabla\Gamma(\vec{x})$ representing the normal to the obstacle surface. $D(\vec{x})$ is a diagonal matrix whose diagonal entries are $\lambda_s, \lambda_1, ..., \lambda_{d-1}$, with

$$\lambda_s = 1 - \frac{1}{\Gamma(\vec{x})}, \tag{C.7}$$

$$\lambda_1, ..., \lambda_{d-1} = 1 + \frac{1}{\Gamma(\vec{x})}. \tag{C.8}$$

each eigenvalue determines the scaling of each direction. Conceptually, as the robot approaches the obstacle, this modulation decreases the velocity for the component in the reference point direction (i.e. toward obstacles) while increases velocity for perpendicular components. The combined effect results in the robot being deflected away tangent to the obstacle surface.

With N obstacles, we compute the modulation matrix $M_i(\vec{x})$ for every obstacle using the procedure above and the individual controllers $\vec{u}_{M_i}(\vec{x}) = M_i(\vec{x}) \cdot \vec{u}(\vec{x})$. The final modulation is the aggregate of all the individual modulations. However, a simple average is insufficient since closer obstacles should have higher influence to prevent collisions.

[86] proposed the following aggregation procedure. Let $\vec{u}_i$ denote the individual modulations, with norms n_i. The final aggregate modulation $\vec{u}$ is calculated as

$$\vec{u} = n_a \vec{u}_a, \tag{C.9}$$

where n_a and $\vec{u}_a$ are the aggregate norm and direction.

The aggregate norm is computed as

$$n_a = \sum_{i=1}^{N} w_i n_i, \tag{C.10}$$

$$w_i = \frac{b_i}{\sum_{j=1}^{N} b_j}, \tag{C.11}$$

$$b_i = \prod_{1 \leq j \leq N, j \neq i} \Gamma_j(\vec{x}). \tag{C.12}$$

The above definition ensures that $\sum_{i=1}^{N} w_i = 1$, and $w_i \to 1$ when $\vec{x}$ approaches $\mathcal{O}_i$ (and only $\mathcal{O}_i$, which holds as long as obstacles are disjoint).

$\vec{u}_a$ is instead computed using what [86] calls "$\boldsymbol{\kappa}$-space interpolation." First, similar to the basis vector matrix $E(\vec{x})$ introduced above, we construct another such matrix, but with respect to the original controller $\vec{x}^* - \vec{x}$. We denote it as $R = [(\vec{x}^* - \vec{x})/||\vec{x}^* - \vec{x}||, \vec{e}_1, ..., \vec{e}_{d-1}]$, where $\vec{e}_1, ..., \vec{e}_{d-1}$ are again orthonomal vectors spanning the null space.

For each $\vec{u}_i$, we compute its coordinate in this new R-frame as $\hat{\vec{u}}_i = R^{-1}\vec{u}_i$. Its $\boldsymbol{\kappa}$-space representation is

$$\boldsymbol{\kappa}_i = \frac{\arccos(\hat{\vec{u}}_i^{(1)})}{\sum_{m=2}^{d} \hat{\vec{u}}_i^{(m)}} \left[\hat{\vec{u}}_i^{(2)}, ..., \hat{\vec{u}}_i^{(d)}\right]^T \in \mathbb{R}^{d-1}, \tag{C.13}$$

where the superscript (m) refers to the m-th entry. $\boldsymbol{\kappa}_i$ is a scaled version of the $\hat{\vec{u}}_i$ with the first entry removed. We perform the aggregation in this $\boldsymbol{\kappa}$-space using the weights w_i calculated above (C.14), transform it back to the R-frame (C.15), and finally transform it back to the original frame (C.16):

$$\boldsymbol{\kappa}_a = \sum_{i=1}^{N} w_i \boldsymbol{\kappa}_i \tag{C.14}$$

$$\hat{\vec{u}}_a = \left[\cos(||\boldsymbol{\kappa}_a||), \frac{\sin(||\boldsymbol{\kappa}_a||)}{||\boldsymbol{\kappa}_a||}\boldsymbol{\kappa}_a^T\right]^T \tag{C.15}$$

$$\vec{u}_a = R\hat{\vec{u}}_a. \tag{C.16}$$

As mentioned in Eq. C.9, the final modulation is $\vec{u} = n_a \vec{u}_a$.

C.4.1 Tail-Effect

An artifact of the above formulation is the "tail-effect," where the robot is modulated to go around the obstacle even when it has passed by the obstacle and the remaining trajectory has no chance of collision under the non-modulated controller. This effect has been observed by [97] for a related but different type of modulation. Fig. C-1, reproduced from the paper by [97] (Fig. 7), shows the tail effect on the left and its removal on the right. This tail effect induces the placement of obstacles at the end

of the "diagonal corridor" as seen in our straight-line deviation experiments (Fig. 5-6, left). If desired, the DS formulation can be modified to remove this effect.

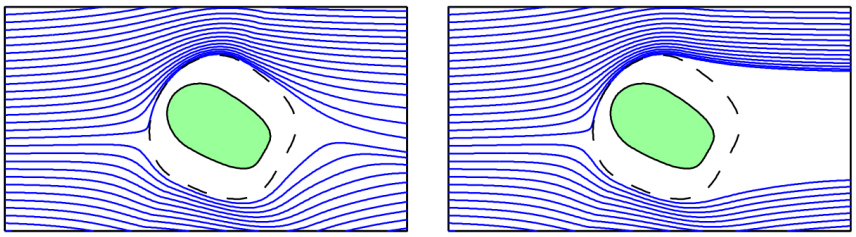


Figure C-1: Tail effect (left) and its removal (right), reproduced from Fig. 7 by [97]. The target is on the far right side.

C.5 RRT Algorithm Description and Sampling

There are many RRT variants with subtle differences. For clarity, Algorithm 2 presents the version that we use.

Algorithm 2: The explicit RRT algorithm used with RoCUS.

Input: Start configuration s_0, target configuration s^*.

$\mathcal{T} \leftarrow$ tree(root $= s_0$);
success $\leftarrow$ attempt-grow($\mathcal{T}$, from $= s_0$, to $= s^*$);
while *not* success **do**
  $s \leftarrow$ sample-configuration( );
  $s_n \leftarrow$ nearest-neighbor($\mathcal{T}$, s);
  success $\leftarrow$ attempt-grow($\mathcal{T}$, from $= s_n$, to $= s$);
  if success **then**
    success $\leftarrow$ attempt-grow($\mathcal{T}$, from $= s$, to $= s^*$);
return path($\mathcal{T}$, from $= s_0$, to $= s^*$)

While RRT is stochastic (unlike DS, IL and RL), the entire randomness is captured by the sequence of C-space samples used to grow the tree, including failed ones. We call this a *growth* $g = [s_1, s_2, s_3, ...]$. The probabilistic completeness property of RRT generally assures that the algorithm will terminate in finite time with probability 1 if a path to the target exists [118]. Thus, hypothetically, given an infinitely long tape containing every entry of g, we can compute a deterministic trajectory $\tau = \text{RRT}(s_0, s^*, g)$ with a finite number of nodes with probability 1.

To enable MH inference, we take inspiration from Bayesian nonparametrics: we instantiate g on an *as-needed* basis. We start with an empty vector of $g = [\,]$. When calculating $\text{RRT}(s_0, s^*, g)$, if a new point beyond existing entries of g needs to be sampled, we append it to g. During MH inference, we use a transition kernel that

operates element-wise on instantiated entries of g (i.e. independently perturbing each entry of g). If the transition kernel does not depend on the current g (e.g. drawing uniformly from the C-space), then past instantiated entries do not even need to be kept.

Note that RRT trajectories are often smoothed *post hoc*. Since our main focus is to evaluate and identify problems for an existing one, we use the original formulation. Moreover, it is easy to use RoCUS to evaluate model updates (e.g. original vs smoothed RRT) as discussed in Sec. 5.7.

C.6 MCMC Sampling Details

We used a truncated Gaussian transition kernel for all experiments. For the RBF-defined 2D environment, we initialize 15 obstacle points with coordinates sampled uniformly in $[-0.7, 0.7]$. The transition kernel operates independently on each obstacle coordinate: given the current value of x, the kernel samples a proposal from $\mathcal{N}(\mu = x, \sigma^2 = 0.1^2)$ truncated to $[-0.7, 0.7]$ (and also appropriately scaled). For the arm reaching task, the target is sampled uniformly from two disjoint boxes, with the left box at $[-0.5, -0.05] \times [-0.3, 0.2] \times [0.65, 1.0]$ and the right box at $[0.05, 0.5] \times [-0.3, 0.2] \times [0.65, 1.0]$. Again, we use the same transition kernel with $\sigma_x = 0.1, \sigma_y = 0.03, \sigma_z = 0.035$ in three directions. Again, the distribution is truncated to the valid target region ($x \in [-0.5, -0.05] \cup [0.05, 0.5], y \in [-0.3, 0.2], z \in [0.65, 1.0]$). In other words, the transition kernel implicitly allows for the jump across two box regions.

In addition, the stochastic RRT controller also requires a transition kernel. As discussed in Sec. 5.5.1, we initialize its values on an as-needed basis. When necessary, we sample a configuration uniformly between the lower- and upper-limit (i.e. $[x_L, x_U]$). For each configuration, the same Gaussian kernel truncated to $[x_L, x_U]$, and $\sigma = 0.1(x_U - x_L)$ is used.

Each sampling run collected 10,000 samples, with the first 5,000 discarded as burn-in. On a consumer-grade computer with a single GeForce GTX 1080 GPU card (for neural network-based controllers), the sampling generally takes around 1 to 3 hours. The number of samples and burn-ins are selected fairly conservatively to ensure representativeness, as Fig. C-2 plots the sampled behavior values in the chain for three analyses and confirms that these numbers are more than sufficient to ensure proper mixing. Note that RoCUS is designed to be an offline analysis tool as opposed to be used for real-time sample generation, and therefore several hours of runtime would be acceptable in most cases. Furthermore, MCMC sampling is embarrassingly parallel by simply using multiple chains concurrently, with the only overhead cost being the discarded burn-in samples.

C.7 2D Environment Details

In this domain, the environment is the area defined as $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. The goal is to navigate from $[x_{\text{start}}, y_{\text{start}}]$ to $[x_{\text{goal}}, y_{\text{goal}}]$. We define a flexible environment

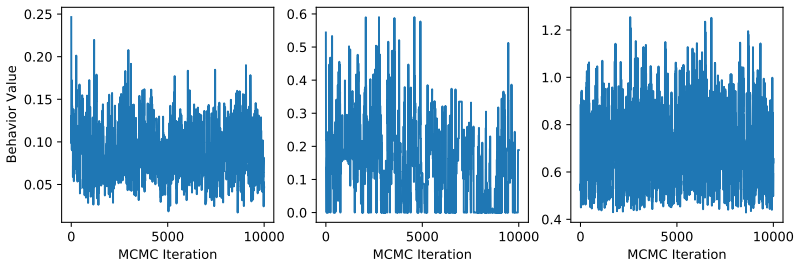


Figure C-2: The sampled behavior values for three MCMC chains. From left to right, the three panels show DS min straight-line deviation on 2D navigation, RRT min straight-line deviation on 2D navigation and RL min end-effector movement on 7DoF arm reaching. The visualization confirms that 10,000 iterations with 5,000 burn-ins are more than sufficient to find representative samples.

representation as a summation of radial basis function (RBF) kernels centered at so-called *obstacle points*. Specifically, given N_O obstacle points $\vec{p}_1, \vec{p}_2, ..., \vec{p}_{N_O} \in \mathbb{R}^2$, the environment is defined as

$$e(\vec{p}) = \sum_{i=1}^{N_O} \exp\left(-\gamma ||\vec{p} - \vec{p}_i||_2^2\right), \tag{C.17}$$

and each point $\vec{p}$ is an obstacle if $e(\vec{p}) > \eta$, for $\eta < 1$ to ensure each obstacle point $\vec{p}_i$ is exposed as an obstacle. Our environments are bounded by $[-1.2, 1.2] \times [-1.2, 1.2]$, and the goal is to navigate from $[-1, -1]$ to $[1, 1]$. $N_O = 15$ and p_i coordinates are sampled uniformly in $x_i, y_i \in [-0.7, 0.7]$. A smaller γ and η makes the obstacles larger and more likely to be connected; we choose $\gamma = 25$ and $\eta = 0.9$. Fig. C-3 shows random obstacle configurations demonstrating high diversity in this environment. We also implement a simple simulator: given the current robot position $[x, y]$ and the action $[\Delta x, \Delta y]$, the simulator clamps $\Delta x, \Delta y$ to the range of [-0.03, 0.03], and then moves the robot to $[x + \Delta x, y + \Delta y]$ if there is no collision, and otherwise simulates a frictionless inelastic collision (i.e. compliant sliding) that moves the robot tangent to the obstacle. Fig. C-3 depicts a randomly selected assortment of 2D environments. These environments demonstrate the flexibility and diversity of the RBF environment definition.

C.8 Implementation Details of 2D Navigation Controllers

C.8.1 IL Controller

The imitation learning controller is a memoryless policy implemented as a fully connected neural network with two hidden layers of 200 neurons each and ReLU activations. The input is 18 dimensional, with two dimensions for the current (x, y) position

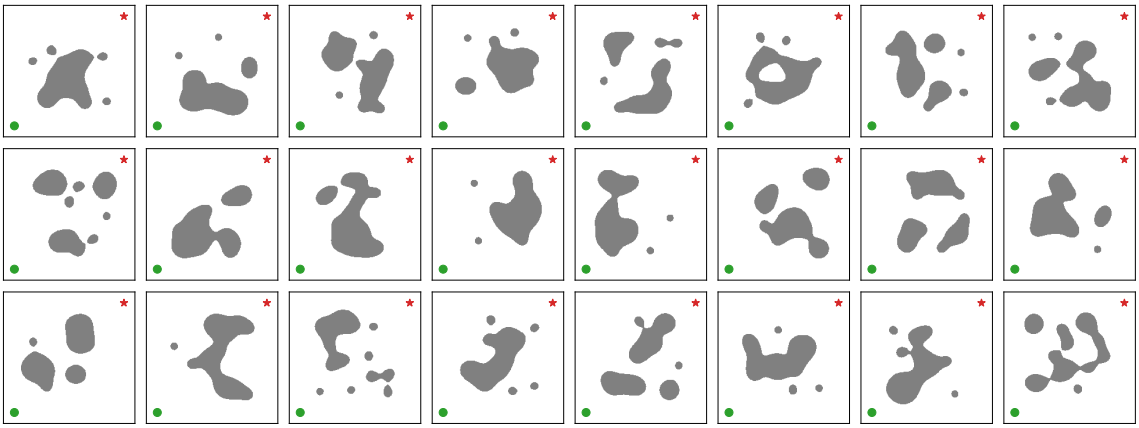


Figure C-3: An assortment of randomly generated RBF 2D environments, providing a sense of the diversity generated with this formulation. The green dots are the environment starting points and the red stars are navigation targets. We show DS modulation for the first three environments in Fig. C-5.

of the robot, and 16 dimensions for a lidar sensor in 16 equally-spaced directions, with a maximum range of 1. The network predicts the heading angle θ, and the controller operates on the action of $[\Delta x, \Delta y] = [0.03\cos\theta, 0.03\sin\theta]$.

The network is trained on smoothed RRT trajectories. Specifically, we use the RRT controller to find and discretize a trajectory. Then the smoothing procedure repeatedly replaces each point by the mid-point of its two neighbors, absent collisions. When this process converges, each point on the trajectory becomes one training data point.

Since only local observations are available and the policy is memoryless, the robot may get stuck in obstacles, which happens in approximately 10% of the runs. In addition, while the output target is continuous, a regression formulation with mean-squared error (MSE) loss is inappropriate, due to multimodality of the output. For example, when the robot is facing an obstacle, moving to either left or right would avoid it, but if both directions appear in the dataset, the MSE loss would drive the prediction to be the average, resulting in a head-on collision. This problem has been recognized in other robotic scenarios such as grasping [215] and autonomous driving [207]. We follow the latter to treat this problem as classification with 100 bins in the $[0, 2\pi]$ range.

C.8.2 DS Controller

For the DS controller, there are two technical challenges in using the modulation [86] on our RBF-defined environment. First, we need to identify and isolate each individual obstacle, and second, we need to define a Γ-function for each obstacle.

To find all obstacles, we discretize the environment into an occupancy grid of resolution 150×150 covering the area of $[-1.2, 1.2] \times [-1.2, 1.2]$. Then we find connected

components using flood fill, and each connected component is taken to be an obstacle.

To define a Γ-function for each obstacle, we first choose the reference point as the center of mass of the connected component. Then we cast 50 rays in 50 equally spaced directions from the reference point and find the intersection point of each ray with the boundary of the connected component. Finally, we connected those intersections in sequence and get a polygon. In case of multiple intersection points, we take the farthest point as vertex of the polygon, essentially completing the non-star-shaped obstacle to be star-shaped, as shown in Fig. C-4.

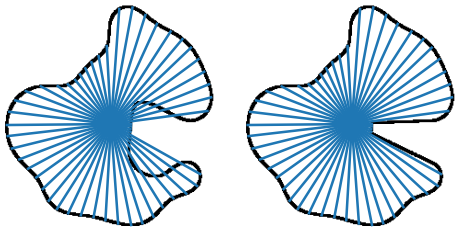


Figure C-4: Left: an obstacle which is not star-shaped. Some radial lines extending from the obstacle's reference point cross the boundary of the obstacle twice. Right: the same obstacle, modified to instead be star-shaped.

Given an arbitrary point $\vec{x}$, we define

$$\Gamma(\vec{x}) = \frac{||\vec{x} - \vec{r}||}{||\vec{i} - \vec{r}||}, \tag{C.18}$$

where $\vec{r}$ is the reference point and $\vec{i}$ is the intersection point with the polygon of the ray from $\vec{r}$ in $\vec{x} - \vec{r}$ direction. It is easy to see that this Γ definition satisfies all three requirements for Γ-functions listed in App. C.4.

Finally, to compensate for numerical errors in the process (e.g. approximating obstacles with polygons), we define the control inside obstacle to be the outward direction, which helps preventing the robot from getting stuck at obstacle boundaries in practice. Three examples of DS modulation of the 2D navigation environment are shown in Fig. C-5.

C.9 Additional Results for 2D Navigation

Legibility We define the instantaneous legibility as the cosine similarity between the current robot direction and the direction to target $\vec{x}^*$, $V(\vec{x}) = \dot{\vec{x}} \cdot (\vec{x}^* - \vec{x})/(||\dot{\vec{x}}|| \cdot ||\vec{x}^* - \vec{x}||)$, with the intuition that a particular run may be confusing to users if the robot does not often align to the target. Though this quantity is bounded by $[-1, 1]$, a general legibility definition may not be. Thus, we use the maximal mode of RoCUS to find DS trajectories and obstacle configurations that achieve *minimal* legibility, by negating $V(\vec{x})$ first. The left two panels of Fig. C-6 present the samples. As expected, most trajectories take large detours due to the presence of obstacles in the center.

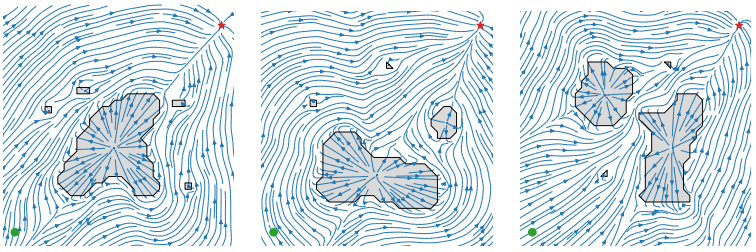


Figure C-5: Streamlines showing the modulation effect of the dynamical system for three 2D navigation tasks. The environments correspond to the first three examples of Fig. C-3. Green dots are starting positions and red stars are navigation targets.

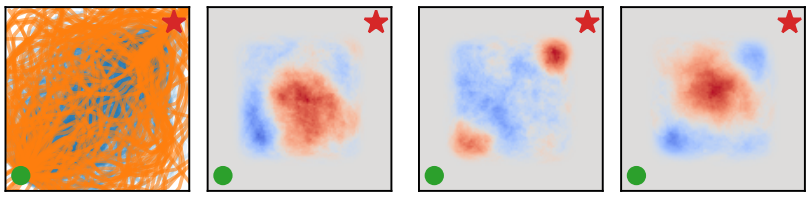


Figure C-6: Left: trajectories and obstacle configurations from sampling minimal DS legibility. Right: obstacle configurations for minimizing and maximizing DS obstacle clearance. These examples show how obstacle positions affect the legibility and clearance behaviors.

Obstacle Clearance We take $V(\vec{x}) = \min_{\vec{x}_o \in \mathcal{O}} ||\vec{x} - \vec{x}_o||$. For the DS, we sample two posteriors to maximize and minimize this behavior. As shown in the right two panels of Fig. C-6, when minimizing obstacle clearance, we see clusters of obstacles in close proximity to the starting and target positions, such that the robot is forced to navigate around them. When maximizing obstacle clearance, we instead see central clusters of obstacles, such that the robot can avoid them by bearing hard left or right.

C.10 Implementation Details of 7DoF Arm Reaching Controllers

C.10.1 RRT Controller

Since the target location is specified in the task space, we first find the target joint space configuration using inverse kinematics (IK). The initial configuration starts with the arm positioned down on the same side as the target. If the IK solution is in collision, we simulate the arm moving to it using position control, and redefine the final configuration at equilibrium as the target (i.e. its best effort reaching configuration). We solve the IK using Klamp't [74].

C.10.2 RL Controller

The RL controller implements the proximal policy gradient (PPO) algorithm [180]. The state space is 22-dimensional and consists of the following:

- 7D joint configuration of the robot,
- 3D position of the end-effector,
- 3D roll-pitch-yaw of the end effector,
- 3D velocity of the end-effector,
- 3D position of the target,
- 3D relative position from the end-effector to the target.

The action is 7-dimensional for movement in each joint, which is capped at $[-0.05, 0.05]$.

Both the actor and the critic are implemented with fully connected networks with two hidden layers of 200 neurons each, and ReLU activations. The action is parametrized as Gaussian where the actor network predicts the mean, and 7 standalone parameters learns the log variance for each of the 7 action dimensions. At test time, the policy deterministically outputs the mean action given a state.

C.10.3 DS Controller

For the DS controller in 7DoF arm reaching, we face the same challenges as in 2D navigation: defining an appropriate Γ-function for the obstacle configuration that holds the three properties introduced by [86] (listed in App. C.4). Additionally, the DS modulation technique does not consider the robot's morphology, end-effector shape, or workspace limits because it only modulates the state of a point-mass. Thus, we implement several adaptations. First, we modulate the 3D position of the tip of the end-effector. The desired velocity of the end-effector tip, given by the modulated linear controller, is then tracked by the 7DoF arm via the same position-level IK solver as the RRT controller.

Second, we used a support vector machine (SVM) to learn the obstacle boundary from a list of points in the obstacle and free spaces, an approach originally proposed by [140]. Then the decision function of the SVM is used as the Γ-function. As shown in Fig. C-7, we discretize the 3D workspace of the robot and generate a dataset of points in the obstacle space as negative class and those in the free space as positive class.

Using the radial basis function (RBF) kernel $K(\vec{x}_1, \vec{x}_2) = e^{-\gamma||\vec{x}_1 - \vec{x}_2||^2}$, with kernel width γ, the SVM decision function $\Gamma(\vec{x})$ has the following form:

$$\Gamma(\vec{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i K(\vec{x}, \vec{x}_i) + b = \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\vec{x} - \vec{x}_i||^2} + b, \tag{C.19}$$

and the equation for $\nabla\Gamma(\vec{x})$ is naturally derived as follows:

$$\nabla\Gamma(\vec{x}) = \sum_{i=1}^{N_{sv}} \alpha_i y_i \frac{\partial K(\vec{x}, \vec{x}_i)}{\partial \vec{x}} = -\gamma \sum_{i=1}^{N_{sv}} \alpha_i y_i e^{-\gamma||\vec{x} - \vec{x}_i||^2} (\vec{x} - \vec{x}_i). \tag{C.20}$$

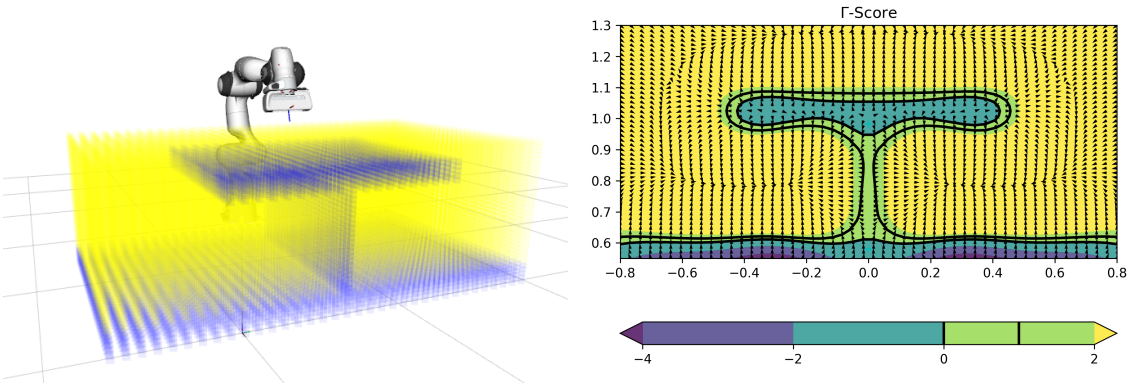


Figure C-7: Left: the division of 3D space as either containing an obstacle or free space. This data is used to train an SVM, which acts as an interpolator. The classification scores of the SVM are used as the Γ function for this 7DoF arm reaching task. Right: a 2D slice showing the smoothed Γ scores.

In Eq. C.19 and C.20, $\vec{x}_i$ $(i = 1, ..., N_{sv})$ are the support vectors from the training dataset, y_i are corresponding collision labels (-1 if position is collided, $+1$ otherwise), $0 \leq \alpha_i \leq C$ are the weights for support vectors and $b \in \mathbb{R}$ is decision rule bias. Parameter $C \in \mathbb{R}$ is a penalty factor used to trade-off between errors minimization and margin maximization. We empirically set the hyper-parameters of the SVM to $C = 20$ and $\gamma = 20$. Parameters α_i and b and the support vectors $\vec{x}_i$ are estimated by solving the optimization problem for the soft-margin kernel SVM using `scikit-learn`. Using this learned Γ-function, Fig. C-8 shows two examples of the modulated trajectory.

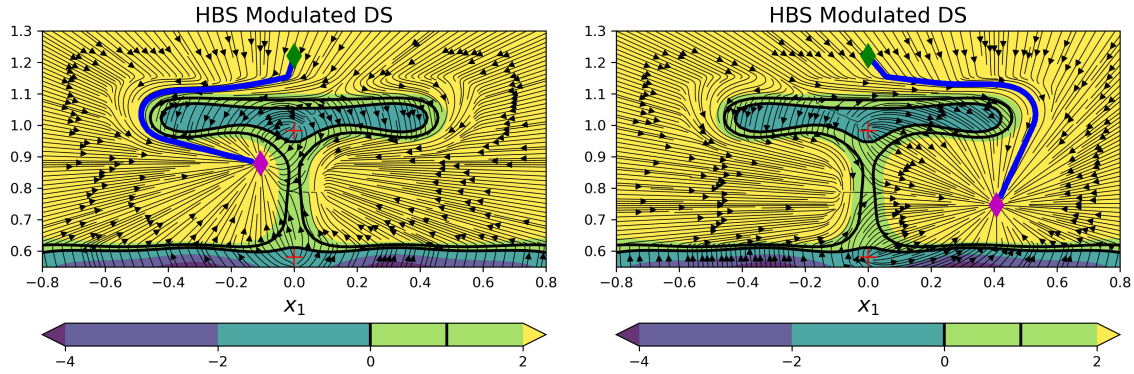


Figure C-8: Cross-sections showing streamlines of the dynamical system modulation effect for two distinct targets in the 7DoF arm reaching task. Red crosses indicate reference points. Green diamond is the initial position of the end-effector for all experiments.

Finally, given a desired modulated 3D velocity for the end-effector tip, $\dot{\vec{x}}_M =$

$\vec{u}_M(\vec{x})$, we compute the next desired 3D position by numerical integration:

$$\vec{x}_{t+1} = \vec{x}_t + \vec{u}_M(\vec{x}_t)\Delta t \tag{C.21}$$

where $\vec{x}_t, \vec{x}_{t+1} \in \mathbb{R}^3$ are the current and next desired 3D position of the tip of the end-effector and $\Delta t = 0.03$ is the control loop time step. $\vec{x}_{t+1}$ is then the target in Cartesian world space coordinates that defines the objective of the position-based IK solver implemented in Klamp't [74].

C.11 Additional Results for 7DoF Arm Reaching

Details on the DS Improvement The DS controller provides guarantees of convergence to a target in the space where modulation is applied (i.e. task-space in our experiments). To adopt this controller for obstacle avoidance with a robot manipulator, [86] simplifies the robot to a spherical shape with center at the end-effector of a 7DOF arm. This translates to considering the robot as a zero-mass point in 3D space but with the boundaries of the obstacles (described by Γ-functions) expanded by a margin with the size of the radius of the sphere.

Since the shape of the Franka robotic hand is rectangular ($6.3 \times 20.7 \times 14$cm) fitting a sphere with the radius of the longest axis will over-constrain the controller and drastically reduce the target regions inside the table dividers. We thus implemented the obstacle clearances by extruding the edges of the top table divider by half of the length of the robot's end-effector (10cm) and the width of the divider by half of the height (7cm). Intuitively, this should be enough clearance to avoid the robot's end-effector colliding with the table dividers. However, when coupling the DS controller with the IK solver to control the 7DoF arm, we noticed that the success rate was below 15%, whereas the success rate is 100% when controlling the end-effector only. We then sampled, via RoCUS, the target locations for the minimal final end-effector distance to target and noticed that all of the successful runs were located on the left-side of the partition (Fig. 5-7 center right).

Since the DS controller approach does not consider collision avoidance in joint-space, in a constrained environment, the robot's forearm or elbow might get stuck on the edges of the table divider—even though the end-effector is avoiding collision. Due to the asymmetric kinematic structure of the robot arm, it is more prone to these situations on the right side of the table divider. Such an insight is not easy to discover as one must understand how the robot will behave in joint space based on its kinematic structure and the low-level controller used (position-based IK). We thus extended the edge extrusions to 20cm. This change improved the controller success rate and behavior drastically as shown in (Fig. 5-7 rightmost).

Legibility We define legibility of reaching to the target on one side of the vertical divider as the average negative distance that the end effector moves in the other direction, $V(\vec{x}) = -\max(\tilde{\vec{x}}_1, 0)$, where $\tilde{\vec{x}}_1 = \vec{x}_1$ if target is on the left, or $\tilde{\vec{x}}_1 = -\vec{x}_1$ otherwise, and $\vec{x}_1$ is the x-coordinate of the robot end effector with right in the

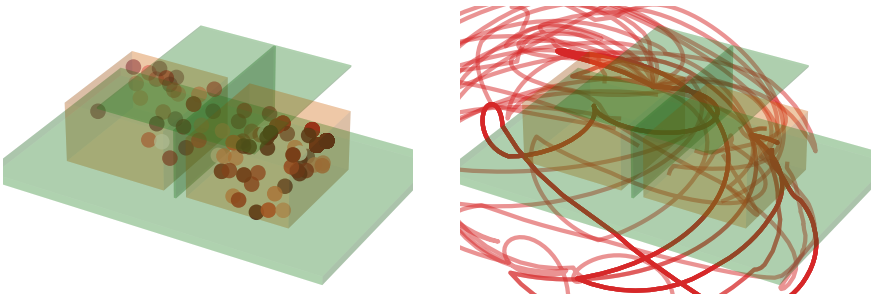


Figure C-9: Posterior samples showing minimal legibility behavior for RRT. RRT is known to exhibit highly illegible behaviors due to its highly stochastic nature, and these samples fit with that expected behavior.

positive direction. We find target locations that are minimally legible and apply the maximal inference mode on the maximum distance measure.

We did not find any illegible motions from RL controllers for 2,000 targets, which is mostly expected since the RL reward is distance to the target. For RRT, however, since we do not use an optimal formulation (e.g., [95, 75]) or perform post-hoc smoothing, the controller is expected to frequently exhibit low legibility. Fig. C-9 plots the posterior target locations and trajectories. The target locations leading to illegible motions are spread out mostly uniformly on the right, but concentrated in far-back area on the left, consistent with our findings on the asymmetry of configuration space. The trajectory plot confirms the illegibility.

C.12 Future Work

There are multiple directions to extend and complement RoCUS for better usability and more comprehensive functionality. First, while we only used RoCUS on individual controllers, future work can readily extend it to *compare two controllers* by defining behavior functions that take in the task and two trajectories, one from each controller, and compute differential statistics. For example, this could be used to find road conditions that lead to increased swerving behavior of a new AV controller, compared to the existing one. Such testing is important to gain a better understanding of *model updates* [16], and is particularly necessary for ensuring that these updates do not unintentionally introduce new problems.

In addition, sometimes it is important to understand particular trajectories sampled by RoCUS. For example, which sensor input (e.g. lidar or camera) is most important to the current action (e.g. swerving)? Why does the controller take one action rather than another (e.g. swerving rather than braking)? Preliminary investigation into this explainable artificial intelligence (XAI) problem in the context of temporally extended decision making has been undertaken [69, 209], but various issues with existing approaches have been raised [13, 214] and future research is needed to address them.

Finally, an important step before actual deployment is to design appropriate user

interfaces to facilitate the two-way communication between RoCUS and end-users. In one direction, the user needs to specify the behavior of interest, and it would be desirable for it to involve as little programming as possible, especially for non-technical stakeholders. In the other direction, RoCUS needs to present the sample visualization, and potentially model explanations as described above, for users to inspect. Here, it is important for the information to be accurate but at the same time not overwhelming.