Journal of Artificial Intelligence Research 67 (2020) 191-234 Submitted 05/2019; published 02/2020

Blind Spot Detection for Safe Sim-to-Real Transfer

Ramya Ramakrishnan RAMYARAM@MIT.EDU
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA 02139

Ece Kamar ECKAMAR@MICROSOFT.COM
Debadeepta Dey DEDEY@MICROSOFT.COM
Eric Horvitz HORVITZ@MICROSOFT.COM
Microsoft Research
14865 NE 36th St, Redmond, WA 98052

Julie Shah JULIE_A_SHAH@CSAIL.MIT.EDU
Massachusetts Institute of Technology
77 Massachusetts Ave, Cambridge, MA 02139

Abstract

Agents trained in simulation may make errors when performing actions in the real world due to mismatches between training and execution environments. These mistakes can be dangerous and difficult for the agent to discover because the agent is unable to predict them a priori. In this work, we propose the use of oracle feedback to learn a predictive model of these *blind spots* in order to reduce costly errors in real-world applications. We focus on blind spots in reinforcement learning (RL) that occur due to incomplete state representation: when the agent lacks necessary features to represent the true state of the world, and thus cannot distinguish between numerous states. We formalize the problem of discovering blind spots in RL as a noisy supervised learning problem with class imbalance. Our system learns models for predicting blind spots within unseen regions of the state space by combining techniques for label aggregation, calibration, and supervised learning. These models take into consideration noise emerging from different forms of oracle feedback, including demonstrations and corrections.

We evaluate our approach across two domains and demonstrate that it achieves higher predictive performance than baseline methods, and also that the learned model can be used to selectively query an oracle at execution time to prevent errors. We also empirically analyze the biases of various feedback types and how these biases influence the discovery of blind spots. Further, we include analyses of our approach that incorporate relaxed initial optimality assumptions. (Interestingly, relaxing the assumptions of an optimal oracle and an optimal simulator policy helped our models to perform better.) We also propose extensions to our method that are intended to improve performance when using corrections and demonstrations data.

1. Introduction

Agents designed to act in the real world are often trained in a simulated environment to learn a policy that can be transferred to a real-world setting. Training in simulation can provide experiences at a low cost, but mismatches between the simulator and the real world can

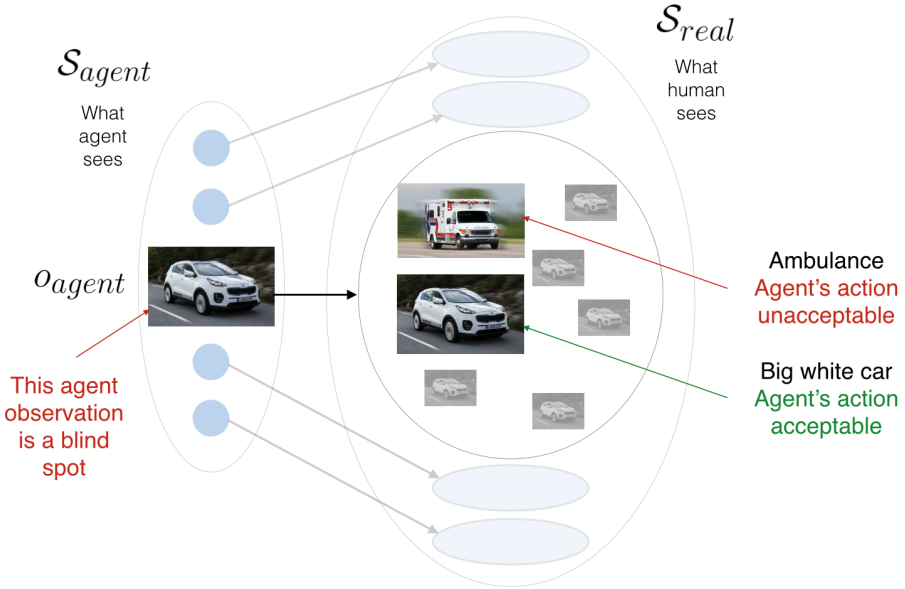


Figure 1: An example of a mismatch between agent representation ($\mathcal{S}_{agent}$) and real-world representation ($\mathcal{S}_{real}$), which can cause agent blind spots. In this case, the agent interprets an ambulance and a big white car as identical and thus continues driving. Because the agent takes an unacceptable action in the case of an ambulance, the corresponding agent observation represents a blind spot.

degrade the learned policy's performance in the open world and lead to costly errors. For example, consider an autonomous driving simulator that includes components for learning how to drive, take turns, stop appropriately, etc., but does not incorporate any emergency vehicles, such as ambulances or fire trucks. When an agent trained in such a simulator is in the presence of an emergency vehicle in the open world, it will continue to drive rather than pull over because it lacks knowledge of the true representation of the world, potentially leading to costly delays or even accidents.

First, let us formally define the problem of discovering blind spots in reinforcement learning (RL). *Blind spots* are regions of the world where agents make unexpected errors due to mismatches between the training environment and the real world. Different kinds of limitations lead to different types of blind spots; in this paper, we focus on blind spots stemming from limitations in state representation. Limitations in the fidelity of the state space result in an agent being unable to distinguish between different real-world states, as highlighted in Figure 1. In the driving example mentioned above, for an agent trained in simulation, states with and without an ambulance would appear the same according to the learned representation. However, the optimal action to take in the open world in these two situations differs significantly, and it would be impossible for an agent that could not distinguish these states from one another to learn to act optimally, regardless of the quantity of simulation-based training.

Such representational incompleteness is ubiquitous in any safety-critical RL application, especially in robotics and autonomous driving, since real-world data can be dangerous

to collect. An expert could make an agent's state representation more complete if the true representation were known a priori, but even with extensive engineering, a gap often exists between simulation and reality. Further, the agent's representation cannot be pre-specified when it is learned automatically through deep reinforcement learning. In cases where complete real-world representation is impossible, it is critical to invest in meta-level reasoning techniques for *identifying* blind spots, which in turn enable representation and policy refinement.

Recent work has increasingly addressed safety with the goal of enabling safer deployment of AI systems. Several works have focused on building more realistic simulators, reducing the gap between simulation and reality (Shah, Dey, Lovett, & Kapoor, 2018b; Dosovitskiy, Ros, Codevilla, Lopez, & Koltun, 2017; Chang, Dai, Funkhouser, Halber, Nießner, Savva, Song, Zeng, & Zhang, 2017). Others (Tobin, Fong, Ray, Schneider, Zaremba, & Abbeel, 2017; Zhang, Leitner, Ge, Milford, & Corke, 2017; Peng, Andrychowicz, Zaremba, & Abbeel, 2018; van Baar, Corcodel, Sullivan, Jha, Romeres, & Nikovski, 2018) have focused on developing more robust training, which involves training on perturbed environments to increase variety in what the agent sees before real-world execution. Cautious exploration in the real world can also enable safe deployment with fewer catastrophic failures during execution (Kahn, Villaflor, Pong, Abbeel, & Levine, 2017; Garcıa & Fernández, 2015). Other recent works have also used human feedback to avoid costly errors (Prakash, Khatwani, Waytowich, & Mohsenin, 2019; Frye & Feige, 2019). While many of these works improve transfer from simulation to the real world, they assume that the agent's representation is sufficient for learning and acting. Our work considers the problem of identifying errors when the agent's representation is incomplete. We propose using human feedback to learn a blind spot model that can improve self-awareness and lead to safer real-world execution.

To discover errors caused by representation limitations, we propose a transfer learning approach that incorporates a learned simulator policy and *limited* oracle feedback to learn where blind spots are likely to occur. The oracle provides information either by performing the task itself (demonstrations) or monitoring and correcting the actions of the agent (corrections), which provides signals to the agent indicating whether its actions were acceptable in each state. We assume that blind spots do not occur at random, and that they correlate with features known to the agent. For example, an agent may lack the features to recognize emergency vehicles, but the existence of emergency vehicles may correlate with observable features, such as vehicle size or color. Under this assumption, we formalize the problem of discovering blind spots as one of supervised learning, where the objective is to learn a blind spot map that provides the likelihood of each agent observation being a blind spot by generalizing predictions to unseen regions of the world. That is, the agent learns the probability that each agent observation corresponds to at least one real-world state in which the agent's chosen action would result in a high-cost error.

Note that learning a predictive model for blind spots is preferred over updating a learned policy when the agent's state representation is insufficient. In the driving example, two states that are indistinguishable to the agent require different actions: a state that includes an ambulance requires the agent to pull over to the side of the road and stop, while a state without an ambulance requires the agent to continue to drive at the speed limit. If the agent updates its policy for both of these similar-appearing states, the consequence could be costly and dangerous. Instead, a blind spot model can be used in any safety-critical

real-world setting where the agent can query an oracle for help in potentially dangerous areas instead of committing to an incorrect and potentially catastrophic action.

Formalizing blind spot discovery as a supervised learning problem introduces several challenges. First, each label received from the oracle provides a noisy signal indicating whether the corresponding agent observation is a blind spot. Since an agent's observation of the world may correspond to multiple real-world states, identifying whether an agent observation is a blind spot requires aggregating multiple labels. In addition, the noise in labels varies across different types of oracle feedback: for example, corrections clearly indicate whether an agent's action is acceptable in a state, whereas demonstrations only depict when the agent's and oracle's behaviors differ. Second, blind spots can be rare, and thus learning about them represents an imbalanced learning problem. Finally, oracle feedback collected through executions (including both corrections and demonstrations) violates the i.i.d. assumption and introduces biases into the learning process.

Our approach leverages multiple techniques to address the problems of noise and imbalance. Prior to learning, we apply expectation maximization (EM) to the dataset of oracle feedback in order to estimate noise in the labels and reduce noise through label aggregation. We also apply oversampling and calibration techniques to address class imbalance. Finally, we experiment with different forms of oracle feedback, including corrections, demonstrations, and randomly-selected states, to quantify the biases in different conditions.

We evaluated our approach using two game domains, and our findings indicate that blind spots can be learned more effectively with our method than baseline approaches, highlighting the benefit of reasoning about different forms of feedback noise. Further, the learned blind spot models are useful for selectively querying an oracle for help during real-world execution. Our evaluations also show that each feedback type introduces different biases that influence the blind spots the agent learns. Overall, corrections are informative, as they provide direct feedback about the agent's policy. The effectiveness of demonstrations varies: in some cases, demonstrations do not account for important errors that the agent may make, resulting in inadequate coverage of all blind spots; while in other cases, demonstration data is sufficient to enable the agent to avoid dangerous regions altogether.

Our initial formulation assumes optimal oracle demonstrations and an optimal simulator policy; we also include an analysis of how our method performs when these assumptions are relaxed. We found that with suboptimal demonstrations, the model can better separate safe and blind spot regions because the distribution of labels from the oracle is more varied. With a suboptimal simulator policy, an agent identifies a larger number of states as blind spots, resulting in a more conservative agent. When both of these assumptions are relaxed, we can obtain equivalent – and occasionally better – performance than when using the original, optimal policies. We also propose data augmentation techniques that improve performance for each feedback type. For corrections, we include an augmentation that propagates human feedback signals to expected future states, which allows an agent to learn about dangerous regions without actually visiting them. For demonstrations, we augment the data with a limited amount of corrections to increase the variety in the data collected.

Our contributions in this work are six-fold: (1) formalizing the problem of discovering blind spots caused by representation incompleteness in reinforcement learning, (2) introducing a transfer learning framework that leverages human feedback to learn a blind spot map of the target world, (3) evaluating our approach across two simulated domains, (4) assessing

the biases of different types of human feedback, (5) analyzing the effect of suboptimal oracle and simulator policies on our approach, and (6) proposing modifications to our method to increase performance with demonstrations and corrections data.

The problem of identifying blind spots was first presented in prior work (Ramakrishnan, Kamar, Dey, Shah, & Horvitz, 2018), but we include several extensions here. First, we relax two of our original optimality assumptions: having access to optimal oracle demonstrations and learning an optimal simulator policy during training. We relax these two assumptions simultaneously (the most realistic case), and demonstrate their complementarity through several experiments. Second, we include additions to our method that improve the prediction of blind spots for both corrections and demonstrations data. In our previous work, we showed that our approach represents a first step toward identifying blind spots, but noted that the approach did not work well in certain settings (Ramakrishnan et al., 2018). The improvements introduced in this paper allow our model to discover blind spots with greater accuracy than our original approach, which can thus lead to improved self-awareness for AI systems. Code for the experiments in this paper is available at `https://github.com/ramya-ram/discovering-blind-spots`.

2. Related Work

Here, we discuss prior work related to the problem of blind spot discovery in reinforcement learning. We first outline work in simulation to real-world transfer that is closely related to our problem; we then discuss relevant work in transfer and lifelong learning, which can involve high-level transfer between different tasks and domains. We also highlight work in supervised learning, in which more robust models are developed to better generalize when training and test distributions differ. Finally, we discuss techniques for outlier and novelty detection.

2.1 Safe Simulation to Real-World Transfer

Reinforcement learning under safety constraints is an active research topic, particularly with regard to transfer from simulation environments to the real world (Garcıa & Fernández, 2015; Munos, Stepleton, Harutyunyan, & Bellemare, 2016; Amodei, Olah, Steinhardt, Christiano, Schulman, & Mané, 2016). Many prior works have focused on either developing realistic simulation environments with more robust training or on more cautious exploration within the real world. While these methods improve robustness and safety, they do not address scenarios in which the agent has a flawed representation that prevents it from learning calibrated uncertainty estimates.

One approach to safe real-world execution is creating more realistic simulators that match the real-world setting as closely as possible. Shah et al. introduced a photorealistic environment called AirSim for autonomous cars and drones (Shah et al., 2018b). CARLA (Dosovitskiy et al., 2017) and TORCS (Wymann, Espié, Guionneau, Dimitrakakis, Coulom, & Sumner, 2000) are car simulators that model different aspects of the real world. Chang et al. introduced a dataset with realistic images for scene understanding, which could potentially be used for then learning how to act in the real world (Chang et al., 2017).

While these simulators provide realistic environments, agents still need to train and learn robust models that perform well in the real world. Domain randomization represents

one method of developing such robust policies (Tobin et al., 2017; Zhang et al., 2017; Peng et al., 2018; van Baar et al., 2018). It involves training the agent on many variants of a task or a wide set of parameters in order to better generalize to a real-world environment. Bousmalis et al. used randomized synthetic data to improve a robot's performance of a grasping task in the real world (Bousmalis, Irpan, Wohlhart, Bai, Kelcey, Kalakrishnan, Downs, Ibarz, Pastor, Konolige, et al., 2018). Several other works have also optimized performance for specific areas in which an agent does not typically perform well, such as adversarial settings and worst-case scenarios (Uesato, Kumar, Szepesvari, Erez, Ruderman, Anderson, Heess, Kohli, et al., 2018; Ruderman, Everett, Sikder, Soyer, Uesato, Kumar, Beattie, & Kohli, 2018).

Another approach to improved real-world generalization uses progressive networks, which allows for sequential task learning (Rusu, Vecerik, Rothörl, Heess, Pascanu, & Hadsell, 2016). In this method, columns are added to the network as new tasks are introduced, preventing the network from performing worse through task-specific fine-tuning. Another method of bridging the reality gap is to learn reusable skills that allow for quick adaptation (He, Julian, Heiden, Zhang, Schaal, Lim, Sukhatme, & Hausman, 2018; Julian, Heiden, He, Zhang, Schaal, Lim, Sukhatme, & Hausman, 2018). He et al. considered a multi-task setting in which a robot used simulation data to learn latent skills and a policy conditioned on this space; the robot was then able to choose a sequence of these learned skills to perform unforeseen tasks in the real world (He et al., 2018). Realistic simulators and robust training enable agents to be more capable of acting in the world, but there is still often a reality gap between simulation and the real world because not every nuance can be modeled or predicted (Ramakrishnan et al., 2018; Ramakrishnan, Kamar, Nushi, Dey, Shah, & Horvitz, 2019).

Another method for safe transfer is cautious exploration during execution. Berkenkamp et al. introduced an algorithm to safely obtain data to learn about the dynamics of a system with theoretical safety guarantees (Berkenkamp, Turchetta, Schoellig, & Krause, 2017). In work by Osband et al., an agent explores based on uncertainty estimates obtained through bootstrapping with random initialization (Osband, Blundell, Pritzel, & Van Roy, 2016). Gal and Ghahramani interpreted dropout in neural networks as equivalent to a Bayesian model, which helps to obtain uncertainty estimates (Gal & Ghahramani, 2016).

In a robotics scenario, these notions of uncertainty can be useful for safer real-world execution. In work by Kahn et al., a robot predicts its uncertainty as it acts within the world, and moves slowly to avoid high-speed collisions, while increasing its velocity within parts of the space where it has greater confidence in its actions (Kahn et al., 2017). Many other works have developed approaches for estimating model uncertainty using various techniques (Lütjens, Everett, & How, 2018; McAllister, Gal, Kendall, Van Der Wilk, Shah, Cipolla, & Weller, 2017; Malinin & Gales, 2018). While these approaches are helpful for avoiding errors in real-world environments, they do not account for circumstances in which the agent has a limited representation of the world and its uncertainty estimates are subsequently misleading. The mistakes that can result from such a scenario will occur in areas where an agent has high confidence but is actually incorrect (blind spots).

2.2 Transfer and Lifelong Learning

Many approaches have improved the transfer of information across tasks (Taylor & Stone, 2009; Pan & Yang, 2010; Barrett, Taylor, & Stone, 2010), as tasks cannot be learned from scratch each time. Transfer learning aims to improve performance of a target task by leveraging knowledge from a source task. While this includes simulation to real-world transfer, it can also refer to transfer across different tasks and domains. Multi-task learning involves optimizing performance over a set of tasks by developing shared knowledge across those tasks. Lifelong learning is a combination of both transfer and multi-task learning that refers to a continual process of learning new tasks while retaining the quality of performance on prior tasks.

Many works in the field of transfer learning have focused on learning mappings between state and action spaces to enable Q-value function or policy transfer (Taylor, Stone, & Liu, 2007; Taylor, Kuhlmann, & Stone, 2008; Ramakrishnan, Zhang, & Shah, 2017). Dai et al. developed a model to learn a mapping by linking the feature space of a source task to the target feature space (Dai, Chen, Xue, Yang, & Yu, 2009). In work by Taylor and Stone, rules are summarized in the source task and transferred to the target (Taylor & Stone, 2007). Several researchers have also considered hierarchical approaches to RL that involve transferring subtasks across domains (Kulkarni, Narasimhan, Saeedi, & Tenenbaum, 2016; Vezhnevets, Osindero, Schaul, Heess, Jaderberg, Silver, & Kavukcuoglu, 2017; Peng, Berseth, Yin, & Van De Panne, 2017; Tessler, Givony, Zahavy, Mankowitz, & Mannor, 2017). Our setup is different from many prior transfer learning scenarios, as they do not reason explicitly about the mismatch in the agent's representation and the true representation of the world (Taylor & Stone, 2009; Barreto, Munos, Schaul, & Silver, 2016; Tobin et al., 2017; Barrett et al., 2010; Christiano, Shah, Mordatch, Schneider, Blackwell, Tobin, Abbeel, & Zaremba, 2016).

Within lifelong learning literature, many works have introduced methods for quick transfer to new, sequentially introduced tasks (Ammar, Tutunov, & Eaton, 2015b; Ammar, Eaton, Luna, & Ruvolo, 2015a; Isele, Rostami, & Eaton, 2016b). Ruvolo and Eaton developed an approach that learns a shared basis, uses this library to transfer to new tasks, and revises the basis as additional tasks are introduced (Ruvolo & Eaton, 2013); this approach performs well on classification and regression problems. Abel et al. used state abstractions to capture important structure within the task to help during transfer (Abel, Arumugam, Lehnert, & Littman, 2018). Brunskill and Li performed a theoretical analysis of when learning options, or temporally extended actions, could be helpful in lifelong learning settings (Brunskill & Li, 2014). In work by Isele et al., transfer was improved considerably by learning relationships between task descriptors and task policies (Isele, Luna, Eaton, Gabriel, Irwin, Kallaher, & Taylor, 2016a).

2.3 Generalization in Supervised Learning

Our work relates to supervised learning literature that trains models to be robust to examples outside of the training distribution. The presence of different training and test distributions during learning can also be referred to as *covariate shift*. One method of addressing this situation is a technique called dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014), which reduces overfitting by dropping units and connections while

training a neural network. Many thinned networks are sampled and trained, and are then combined into a single network at test time. This process results in better generalization than use of neural nets without dropout across several classification datasets.

Another way to train more robust models that do not overfit is to use adversarial examples (Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, & Fergus, 2013; Kurakin, Goodfellow, & Bengio, 2016). Shaham et al. developed a robust optimization approach in which a network is optimized over several perturbed examples in an uncertainty set, $\mathcal{U}$ (Shaham, Yamada, & Negahban, 2015). Athalye et al. generated a more informative set of adversarial examples based on transformations of physical objects, which identifies 3D adversarial objects (Athalye & Sutskever, 2017). While these approaches to generating examples increase the robustness of the learned model, they do not help when the set of possible examples is incomplete. In work by Tramèr et al., models are trained against *black-box* adversaries that have no information about the model's inner workings, rather than *white-box* adversaries that have full knowledge of the algorithm and model parameters (Tramèr, Kurakin, Papernot, Goodfellow, Boneh, & McDaniel, 2017).

While adversarial examples represent one way of identifying model errors, it is useful for machine learning systems to also be self-aware and actively query for help in regions where it is likely to make incorrect predictions (Cohn, Ghahramani, & Jordan, 1996; Zhu, Wang, Tsou, & Ma, 2009; Zhang, Deng, Marchi, & Schuller, 2013; Kapoor, Grauman, Urtasun, & Darrell, 2007; Joshi, Porikli, & Papanikolopoulos, 2009; Roy & McCallum, 2001). One useful technique for active learning is estimating model uncertainty, which can guide the querying process. Uncertainty quantification in deep learning (Gal, 2016; Tripathy & Bilionis, 2018; Kendall, Badrinarayanan, & Cipolla, 2015) can be extremely beneficial for learning high-error regions of the model for future refinement.

However, traditional active learning does not directly apply when it is difficult to construct or identify these examples before deploying a model in the open world. Even with extensive adversarial training, a gap can still exist between the data used for training the model and the true distribution of data. Lakkaraju et al. introduced a method for identifying regions of unknown unknowns in discriminative classifiers when deployed into the open world (Lakkaraju, Kamar, Caruana, & Horvitz, 2017). In this approach, data points are clustered in an unsupervised manner, followed by a multi-arm bandit algorithm (with each cluster representing an arm) to efficiently identify regions of the feature space in which the classifier is most likely to make mistakes. It is not straightforward to apply this approach to RL, however, because examples (states) are no longer i.i.d., as in supervised learning. In RL, states are visited according to a distribution induced by executing either the learned policy or the oracle's policy. There can also be multiple acceptable labels (actions) for each state, rather than a single "correct" label. Finally, certain mistakes in the real world can be catastrophic, necessitating risk-sensitive classification to prioritize identifying rare blind spots (Zadrozny, Langford, & Abe, 2003).

2.4 Novelty and Outlier Detection

As indicated in previous work, outlier and novelty detection are related but not directly applicable to blind spot discovery (Chandola, Banerjee, & Kumar, 2009; Gupta, Gao, Aggarwal, & Han, 2014; Campos, Zimek, Sander, Campello, Micenková, Schubert, Assent, &

Houle, 2016). Outlier detection aims to identify instances within the training data that are far from the rest of the examples. Novelty detection assumes that the training data does not contain outliers, and instead involves identifying whether a new instance not seen in training deviates significantly from the training set. One reason blind spots differ from outliers is that they are not rare instances, but systematic regions within the agent's representation where the training environment does not match the testing environment, leading to costly errors. In our work, we present a method for efficiently identifying these regions through oracle feedback. Another difference between outliers and blind spots is that detecting outliers requires access to the correct set of features in order to distinguish novel examples (Markou & Singh, 2003a, 2003b); in our work, the available features are insufficient for differentiating blind spots from safe regions.

Several prior methods have used kernel techniques to identify outliers (Schölkopf, Williamson, Smola, Shawe-Taylor, & Platt, 2000; Hoffmann, 2007; Campbell & Bennett, 2001; Bodesheim, Freytag, Rodner, Kemmler, & Denzler, 2013). Hoffman introduced a novelty measure by using kernel PCA to map points into a higher-dimensional space. The system performed PCA to obtain independent components and computed the reconstruction error based on this space (Hoffmann, 2007). Bodeshim et al. mapped training examples with a specific class into one point (Bodesheim et al., 2013); this projected subspace has zero variance within each class and can be used to predict the novelty of new instances through a distance measure.

In work by Blouvshtein and Cohen-Or, outliers were identified by reasoning about distances between data points (Blouvshtein & Cohen-Or, 2018). A given distance is predicted to be an outlier if it breaks the triangle inequality for many triangles. While many of these methods rely upon distances to detect outliers, one challenge is that many examples look like outliers in high-dimensional spaces. Radovanović et al. demonstrated that when attributes are not noisy, outliers can be identified as more pronounced within the high-dimensional space (Radovanović, Nanopoulos, & Ivanović, 2015). However, these works are, again, conditioned upon the availability of features to identify novel or irregular instances, which we do not assume in our own work.

3. Problem Formulation

We formulate the discovery of agent blind spots in reinforcement learning as a transfer learning problem, in which an agent is trained in a simulation environment M_{agent_sim} and is deployed and evaluated in a real-world environment M_{real}. The simulator is modeled as a Markov Decision Process (MDP): $M_{agent\_sim} = \{\mathcal{S}_{agent}, \mathcal{A}, T_{sim}, R_{sim}\}$. The state space $\mathcal{S}_{agent}$ defines all possible states in the simulation world. The representation of this state space can be manually designed or learned. The action space of the agent is defined as $\mathcal{A}$. The transition function $T_{sim} : \mathcal{S}_{agent} \times \mathcal{A} \times \mathcal{S}_{agent} \rightarrow \mathbb{R}$ models the dynamics in the simulation environment, which is an approximation of the real world dynamics based on the agent's representation $\mathcal{S}_{agent}$. The reward function $R_{sim} : \mathcal{S}_{agent} \times \mathcal{A} \rightarrow \mathbb{R}$ specifies the reward the agent receives for taking an action from a specific state. The agent trains in this world and learns a policy $\Pi_{sim} : \mathcal{S}_{agent} \rightarrow \mathcal{A}$, which maps states in $\mathcal{S}_{agent}$ to actions $\mathcal{A}$.

The real-world environment is defined as a Partially-Observable Markov Decision Process (POMDP) because the agent does not observe the true state of the world and instead

receives an observation of this true state: $M_{real} = \{\mathcal{S}_{real}, \mathcal{A}, T_{real}, R_{real}, \Omega, O\}$. In this environment, $\mathcal{S}_{real}$ defines all possible real-world states, and $\Omega = \mathcal{S}_{agent}$ defines the space of all possible agent observations. The agent observes states of the world through its representation defined in simulation. While the state space $\mathcal{S}_{agent}$ was sufficient for learning in the simulator, it lacks important features for acting safely in the real world so the agent's observation space is a subset of the real-world state space $\mathcal{S}_{agent} \subset \mathcal{S}_{real}$. The observation function $O : \mathcal{S}_{real} \times \mathcal{A} \rightarrow \mathcal{S}_{agent}$ is a deterministic mapping between the true state and the agent's observation of that world based on $\mathcal{S}_{agent}$. The action space $\mathcal{A}$ is identical to that in training. The true transition and reward functions T_{real} and R_{real} in the real world are based on $\mathcal{S}_{real}$ rather than $\mathcal{S}_{agent}$, so these functions do not match the approximations in simulation.

For each real-world state $s_{real} \in \mathcal{S}_{real}$, the agent receives a flawed observation of the world in terms of its representation $o_{agent} \in \Omega = \mathcal{S}_{agent}$. Because $\mathcal{S}_{agent} \subset \mathcal{S}_{real}$, multiple states look identical to the agent, also known as perceptual aliasing (Kaelbling, Littman, & Moore, 1996). The agent has access to an oracle or human that observes the true state of the world $s_{real} \in \mathcal{S}_{real}$. The agent can only reason within the agent's observation space $\mathcal{S}_{agent}$, while the oracle has access to the true real-world state space and provides feedback based on $\mathcal{S}_{real}$.

Our goal is to use the agent's learned policy from simulation and a limited budget, B, of feedback from an oracle, O, in order to learn a blind spot model of the real world that indicates the probability that each agent observation is a blind spot $\mathcal{S}_{agent} \rightarrow [0, 1]$. This learned model can then be used during real-world task execution to query a human for help at states with a high probability of being blind spots. Prior works have investigated the use of human feedback for guiding agents in RL tasks (Argall, Chernova, Veloso, & Browning, 2009; Knox & Stone, 2009; Saunders, Sastry, Stuhlmueller, & Evans, 2017; Christiano, Leike, Brown, Martic, Legg, & Amodei, 2017; Griffith, Subramanian, Scholz, Isbell, & Thomaz, 2013), but we use oracle feedback to learn a blind spot model of the real world rather than to learn a policy.

An oracle, $O = \{A(s, a), \pi_{real}\}$, is defined by two components: an acceptable function $A(s, a)$ and an optimal real-world policy π_{real}. The acceptable function $A(s, a)$ provides direct feedback about the agent's actions by returning 0 if action a is acceptable in state s, and 1 otherwise. In our experiments, we simulated $A(s, a)$ by defining acceptable actions as those with values within some δ of the optimal action value for that state; however, $A(s, a)$ can be defined in many ways. The optimal policy π_{real} is used when the oracle is providing demonstrations within the real world. In practice, oracles can be humans or other agents with more expensive and/or complementary mode sensors (e.g., lidar cannot see color but can sense 3D shapes, while cameras can detect color but not 3D shapes).

An observation in the agent's representation $o_{agent} \in \mathcal{S}_{agent}$ is defined as a blind spot ($B(o_{agent}) = 1$) if:

$$\exists s_{real} \in \mathcal{S}_{real} \text{ s.t. } A(s_{real}, \pi_{sim}(o_{agent})) = 1. \tag{1}$$

In other words, blind spots are observations in the agent's representation where the agent's learned action is unacceptable in at least one real-world state that maps to it. Intuitively, if two states appear identical to the agent, and that agent is taking an unacceptable action within either of them, it should mark the flawed state it observes as a blind spot.

The agent's objective is to use the learned policy π_{sim} and a limited budget of B labels from the oracle $O = \{A(s, a), \pi_{real}\}$ in order to learn a blind spot model, $M = \{C, t\}$. The classifier $C : \mathcal{S}_{agent} \rightarrow \Pr(B(\mathcal{S}_{agent}) = 1)$ predicts, for each agent observation $o_{agent} \in \mathcal{S}_{agent}$, the probability that o_{agent} is a blind spot in the real-world environment, while the probability threshold t specifies the cutoff for classifying an agent observation as a blind spot.

3.1 Agent Observations

A perfect label for learning the blind spot map would be the tuple $< o^i_{agent}, l^i_p >$, such that l^i_p is 1 when o^i_{agent} is a blind spot – a real-world state corresponding to o^i_{agent} exists where $\pi_{sim}(o^i_{agent})$ is not acceptable – and l^i_p is 0 otherwise. Since the oracle and agent have different representations of the world, the oracle's actions are based on $\mathcal{S}_{real}$, not $\mathcal{S}_{agent}$. Thus, labels collected from the oracle are associated with *state representation (SR) noise* as the agent and the oracle perceive the world differently. In addition, some forms of oracle feedback may provide weaker information about the quality of an agent's actions: instead of providing information about whether an action is acceptable, the feedback may indicate when the agent's and oracle's actions differ, referred to as *action mismatch (AM) noise*. We describe both types of noise in detail below.

3.1.1 State Representation Noise

State representation (SR) noise occurs because the agent and oracle are operating within two different representations. The agent's representation is limited, and has missing features that cause the agent to observe many distinct real-world states as identical to one another. When the oracle provides a label about a real-world state, the agent cannot disambiguate this label from labels provided for other real-world states that look identical and map to the same agent observation, resulting in state representation noise.

Let s^i_{real} be a real-world state and o^i_{agent} be the agent observation s^i_{real} corresponds to. A label tuple from the perspective of the oracle is defined as a tuple, $< s^i_{real}, l^i_a >$, where $l^i_a \in \{0, 1\}$ is the resulting label such that $l^i_a = A(s^i_{real}, a^i_{agent})$ and $a^i_{agent} = \pi_{sim}(o^i_{agent})$. However, due to a limited state representation, the label tuple from the agent's perspective is $< o^i_{agent}, l^i_a >$. When many real-world states map to the same agent observation in this manner, the agent receives several noisy labels for a single observation. For example, if the agent takes an acceptable action in one real-world state and an unacceptable action in a different state that the agent interprets as identical to the first, it will receive two labels for this agent observation: a 0 and a 1.

To return to our definition of blind spots: if the agent receives even one unacceptable label for any real-world state, the corresponding agent observation is automatically identified as a blind spot. Thus, receiving an unacceptable label is a perfect signal that the corresponding agent observation is a blind spot. However, if the agent receives many acceptable labels, the agent cannot mark the observation as "safe" (i.e., not a blind spot) because there may be other real-world states that map to this one where the agent's action would be unacceptable. The main property of SR noise is that receiving many "safe" labels does not guarantee that the corresponding agent observation is safe.

3.1.2 Action Mismatch Noise

The formulation described thus far assumes that the oracle provides direct feedback about the agent's actions using the acceptable function $A(s, a)$. Another form of feedback allows the oracle to provide demonstrations using π_{real} while the agent simply observes, which might be of lower cost to the oracle than directly monitoring and correcting the agent's actions. In this case, when feedback about the agent's actions is not given directly, an additional form of noise is introduced: action mismatch (AM) noise. If the oracle takes action a_i but the agent planned to take a_j, this could be indicative of a blind spot because the agent is not following the optimal action. However, two actions can both be acceptable in a state, so a mismatch does not necessarily imply that the given agent observation is a blind spot. The main property of AM noise is that noisy unacceptable labels are generated for safe regions, and the agent should reason about this noise to avoid being overly conservative and labeling safe areas as blind spots.

4. Approach

We now present a framework for learning blind spots in RL, as depicted in Figure 2. The pipeline includes a data collection phase, during which the agent receives data from an oracle through various forms of feedback. Since each feedback type is associated with noise, we introduce a label aggregation step that estimates the noise in the labels using EM and predicts the true label of each visited agent observation through aggregation of labels. To generalize observations from visited agent observations to unseen regions, we use supervised learning. This learning step assumes that blind spots do not occur at random, but instead correlate with existing features that the agent has access to.

Since blind spots are typically rare within data, learning about them represents an imbalanced learning problem. To address this, we first oversample the blind spot examples and then perform calibration in order to correct estimates of the likelihood of blind spots. Calibrated estimates are important for our domain, as they can be used to decide whether to request oracle help. This allows the agent to accurately trade off the likelihood of error with the cost of querying the oracle in execution.

4.1 Data Collection

In order to develop a system that can learn blind spots in a real-world environment, we first must collect data from an oracle. This oracle can either provide feedback about the agent's actions directly through the acceptable function $A(s, a)$ or provide a demonstration of the optimal action, which the agent can observe, using π_{real}. We now discuss specific forms of oracle feedback and the type of noise (SR and/or AM) they each contain.

4.1.1 Corrections (C)

The first feedback type is corrections (C), in which the oracle monitors and corrects the actions of the agent as it acts within the real world. As shown in Figure 3, the process for data collection is as follows: The agent first visits a state $s^i_{real} \in \mathcal{S}_{real}$ in the real world based on its simulator policy, π_{sim}. The agent receives an observation of this state o^i_{agent} and computes the action it plans to take, $a^i_{agent} = \pi_{sim}(o^i_{agent})$. The agent then obtains a

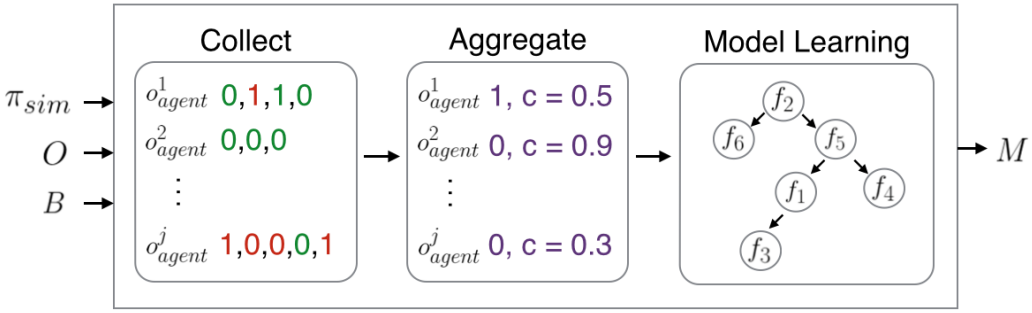


Figure 2: The full pipeline of our approach. To learn a blind spot model, the agent first collects data from an oracle using one of the feedback types, then aggregates noisy labels in order to predict whether each agent observation is a blind spot. Finally, a classifier is trained with this data to generalize over the full space of agent observations.

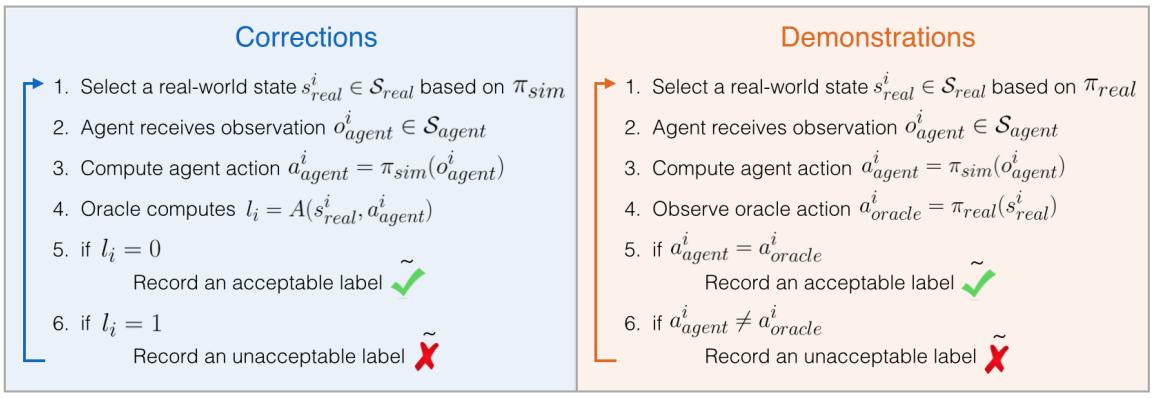


Figure 3: This figure outlines the process of data collection with corrections and demonstrations serving as feedback methods. With the use of corrections, the agent receives direct feedback about its own actions through the acceptable function; with demonstrations, the agent obtains noisy signals of acceptability by observing action matches and mismatches.

feedback label, $l_i = A(s^i_{real}, a^i_{agent})$, from the oracle at that state, where $l_i = \{0, 1\}$. The agent records an acceptable label if $l_i = 0$, and an unacceptable label if $l_i = 1$. Given that many real-world states appear identical to the agent and thus map to the same agent observation, the resulting dataset, $D = \{(o_{agent}, [l_1, ..., l_k])\}$, has many noisy labels for each agent observation due to state representation (SR) noise. (Corrections contain no AM noise because the oracle directly provides feedback about the agent's actions through the acceptable function.)

4.1.2 Demonstrations-Action Mismatch (D-AM)

Corrections can be expensive to obtain because doing so involves constant monitoring of the agent. Demonstrations represent an alternative form of feedback that involves the

oracle acting in the real world using π_{real}. Figure 3 depicts the process of obtaining data from demonstrations. In this case, the oracle selects states based on the policy π_{real} in the real world. The agent receives an observation o^i_{agent} of the chosen real-world state s^i_{real} and computes the action it would take given that observation $a^i_{agent} = \pi_{sim}(o^i_{agent})$. The agent then observes the oracle's action, $a^i_{oracle} = \pi_{real}(s^i_{real})$. In this feedback type, the agent does not learn about the acceptability of its own action; instead, it compares its action to that of the oracle's and uses action matches and mismatches as proxies for acceptability. If the agent's action matches the oracle's action, $a^i_{agent} = a^i_{oracle}$, the agent knows its action is acceptable to perform in the real world. However, if the actions do not match $a^i_{agent} \neq a^i_{oracle}$, this does not necessarily mean that the agent performed an unacceptable action, as multiple acceptable actions may exist for that state. The agent notes a noisy unacceptable label for all action mismatches. This condition results in a dataset, $D = \{(o_{agent}, [l_1, l_2, ..., l_k])\}$, with noisy unacceptable labels; thus, demonstrations-action mismatch (D-AM) includes both SR noise and AM noise.

4.1.3 Demonstrations-Acceptable (D-A)

Corrections and demonstrations-action mismatch differ in two ways: distribution of data and AM noise. We want to decouple these effects; thus, we include a condition without AM noise. This feedback type, demonstrations-acceptable (D-A), is collected similarly to D-AM and followed by a review period in order to get direct feedback about the agent's mismatched actions. For all states with action mismatches, the agent queries the oracle function, $A(s^i_{real}, \pi_{sim}(o^i_{agent}))$, to resolve action mismatch ambiguities. Thus, all noisy unacceptable labels become either "safe" labels (because the agent's action, while different from the oracle's, is acceptable) or true "unacceptable" labels (which confirm that the given action is unacceptable). While AM noise is resolved in this condition, SR noise still exists. The only difference between D-A and C is that D-A generates states according to π_{real}, while C generates states according to π_{sim}.

4.1.4 Random-Acceptable (R-A)

Demonstrations and corrections are forms of feedback that allow a human to naturally provide information to an agent (Christiano et al., 2017). However, the visited states are biased according to the policy that generated them; furthermore, collecting data based on policies violates the i.i.d. assumption of supervised learning. To evaluate the effect of this bias, we include two baseline approaches in which states are randomly selected. In the random-acceptable (R-A) condition, states are randomly sampled in the real world. For each state, the oracle provides a label using the acceptable function, and this label is recorded. The data collection process is identical to D-A, except that states are now chosen randomly.

4.1.5 Random-Action Mismatch (R-AM)

In the random-action mismatch (R-AM) condition, states within the real world are randomly sampled, and the oracle computes its action for a given state based on π_{real}. The agent records an action match as an acceptable label, and a mismatch as a noisy unacceptable label. R-AM is identical to D-AM, except for the random sampling of states.

Table 1: This table outlines the feedback types we consider in this work. We indicate the types of noise each contains and whether each feedback can contain data bias.

Feedback Type	SR Noise	AM Noise	Bias
Corrections (C)	✓	-	✓
Demonstrations-action mismatch (D-AM)	✓	✓	✓
Demonstrations-acceptable (D-A)	✓	-	✓
Random-action mismatch (R-AM)	✓	✓	-
Random-acceptable (R-A)	✓	-	-

4.1.6 Summary of Feedback Types

All feedback types result in a dataset, $D = \{(o_{agent}, [l_1, ..., l_k])\}$, which includes a set of noisy acceptable and unacceptable labels for each agent observation. For feedback types lacking AM noise, when an unacceptable label is given by the oracle for any real-world state, the matching agent observation is marked as a blind spot. In other words, for each $(o_{agent}, [l_1, ..., l_k]) \in D$, if $\exists l_i = 1$, o_{agent} is a blind spot. For an agent observation, if all corresponding labels are acceptable, the agent should reason about the likelihood of the observation being a blind spot or safe based upon the number of labels collected. If the feedback type contains AM noise, the agent must reason about noisy unacceptable labels when learning to identify blind spots.

D-AM is the most difficult type of feedback to reason about because it includes SR noise, AM noise, and bias, while corrections represents the most informative feedback type because it provides direct feedback about the agent's policy. However, we expect demonstrations to be easier to obtain than corrections because the oracle simply acts in the world according to its policy. In Section 6, we discuss the biases and tradeoffs of using demonstrations versus corrections data and how both of these compare to the baselines that involve randomly sampled data. Table 1 summarizes all feedback types and the noise/bias observed in each.

4.2 Aggregating Noisy Labels

Across all conditions, the collected data has many noisy labels that must be aggregated. Through data collection, we obtained a dataset of noisy labels, $D_n = \{(o_{agent}, [l_1, l_2, ..., l_k])\}$, for regions the agent has visited; these labels are noisy due to SR and AM noise. The model must reason about these noisy labels obtained from the oracle in order to determine whether the true label of the agent observation is "blind spot" or "safe."

For label aggregation, we use the Dawid-Skene algorithm (DS), which is a popular approach for addressing label noise in data collection (Dawid & Skene, 1979). We prefer this approach because it has a small number of parameters to estimate, works well over sparse data sets, and has been shown to consistently work well across problems (Sheshadri & Lease, 2013). DS takes a dataset of noisy labels as input, and its goal is to predict the true labels of instances by estimating the prior distribution of the data (the ratio of blind spots vs. safe regions) and the confusion matrix, which is the noise model between noisy labels (acceptable vs. unacceptable) and true labels (blind spots vs. safe). The algorithm is unsupervised; it uses EM in its estimation.

Algorithm 1 Dawid-Skene

Initialize: $\hat{L}_{ij} = \frac{n_{im}}{\sum_{m=1}^{N_c} n_{im}}$

while not converged **do**

M step:

$$\Psi_{jm} = \frac{\sum_{i=1}^{|D_n|} \hat{L}_{ij} n_{im}}{\sum_{m=1}^{N_c} \sum_{i=1}^{|D_n|} \hat{L}_{ij} n_{im}}$$

$$\alpha_j = \frac{\sum_{i=1}^{|D_n|} \hat{L}_{ij}}{|D_n|}$$

E step:

$$\hat{L}_{ij} = \frac{\prod_{m=1}^{N_c} (\Psi_{jm})^{n_{im}} \alpha_j}{\sum_{q=1}^{N_c} \prod_{m=1}^{N_c} (\Psi_{qm})^{n_{im}} \alpha_q}$$

end while

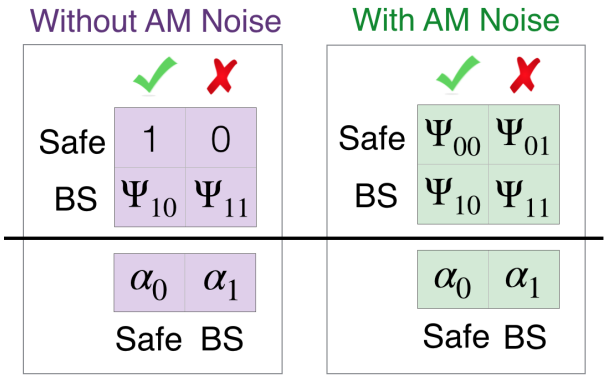


Figure 4: This figure depicts the way in which we constrain the Dawid-Skene algorithm for feedback types with and without AM noise. When there is no AM noise, the confusion matrix can be constrained during learning, but there is no such structure when the data collected includes AM noise.

Algorithm 1 details the full approach. Assume that $\hat{L}_{ij}$ represents the probabilities that each datapoint in the data, $i \in [1, 2, ..., |D_n|]$, is in each class: $j \in [1, N_c]$, where N_c is the number of classes. Because we are interested in labelling states as "safe" or "blind spot", we have $N_c = 2$ classes. For example, if $\hat{L}_{3,2} = 0.6$, the 3rd state in the data is predicted to be a blind spot with 0.6 probability and safe with 0.4 probability. The prior of safe vs. blind spot agent states is denoted as α_j where $j \in [1, .., N_c]$ and the confusion matrix is Ψ_{jm} where $j, m \in [1, .., N_c]$. First, the true labels of instances are initialized by averaging the labels for each agent observation. Then, in the M step, the confusion matrix and prior are estimated based on the initialized true labels. Using these quantities, the true labels are re-estimated in the E step by weighting based on the estimated noise. The algorithm iterates until convergence.

Different forms of feedback are associated with particular noise types that can be used to inform the aggregation approach. For example, for all feedback types without AM noise (C, D-A, R-A), safe regions never receive unacceptable labels because when an agent observation is truly safe for the agent, all corresponding real-world states will also be safe. Also, since the agent receives direct signals based upon its actions, it will not receive noisy labels from action mismatches. We can modify DS to leverage this property by constraining the confusion matrix that DS attempts to estimate. Specifically, one row of the confusion matrix (corresponding to safe regions) can be constrained to have 100% acceptable labels and 0% unacceptable labels. The left-hand side of Figure 4 depicts the parameters that DS must learn for feedback types without AM noise: the ratio of acceptable and unacceptable labels for blind spot regions and the prior distribution of safe and blind spot regions.

In the presence of AM noise, the agent can take a different action from the oracle while remaining safe; we denote this action mismatch as a noisy unacceptable label. As both

safe and blind spot regions can now receive either label ("acceptable" or "unacceptable"), there is no structure within the data to constrain the noise estimation problem. Thus, for feedback types with AM noise (D-AM, R-AM), we use the original DS algorithm to learn all parameters, as shown in Figure 4. The output of aggregation is a dataset, $D_a = \{(s = o_{agent}, \hat{l}, c)\}$, where $\hat{l}$ represents the estimated true label and c is the associated confidence.

4.3 Model Learning

With estimated true labels, we train a supervised learner to predict which observations in the agent's representation are likely to be blind spots in the real world. Due to the relative rarity of blind spot regions, the major challenge in model learning is class imbalance. With only a few blind spots, the model will learn to predict all regions as safe, which can be extremely dangerous. To deal with class imbalance, we oversample blind spot instances to generate balanced classes in the training data, and then calibrate the model to provide better estimates.

The full model-learning process is as follows: a random forest (RF) classifier is trained with data from aggregation $D_a = \{(o_{agent}, \hat{l}, c)\}$, weighted according to c. The output is a blind spot model $M = \{C, t\}$, which includes a classifier C and a threshold t. To learn M, the system performs a randomized hyperparameter search over RF parameters, runs three-fold cross-validation with oversampled data for each parameter configuration, and obtains an average F1-score. The hyperparameters with the highest average F1-score are chosen to train the final model.

For the final training round, we reserve 30% of the full training data for calibration. We oversample the rest of the data and train an RF classifier using the best parameters. In order to calibrate the model after training on oversampled data, we vary the threshold that specifies a cutoff probability for classifying an agent observation as a blind spot. For each possible threshold t, the system measures the percentage of blind spots predicted by the model on the held-out calibration data, and chooses t such that the prior of blind spots on the calibration set matches the prior in the training data, as estimated by the DS approach. The final output, $M = \{C, t\}$, includes the learned RF classifier, C, and the threshold, t.

5. Experimental Setup

In order to evaluate our system's ability to identify blind spots, we conducted experiments within two domains. For each domain, we used one version of the domain to train the agent and a modified version to simulate a real-world setting that did not match the training scenario.

5.1 Domains

The first domain is a modified version of the game Catcher, in which the agent is required to catch falling fruits. In Catcher, a fruit starts from the top of the screen at a random x location, then falls straight down at a constant speed. The agent controls a paddle at the bottom of the screen that can remain stationary, move left, or move right. The state of the game in the simulation environment is represented as $[x_p, x_f, y_f]$, where x_p is the x location of the player and (x_f, y_f) represents the fruit's location. In simulation, the reward

is proportional to the player's x distance away from the fruit, or $W - |x_p - x_f|$, where W represents the width of the screen.

For this domain, the real world is split into two regions: the left-hand side, which appears exactly like simulation, and the right-hand side, which has a probability p of being like simulation, and a probability $1 - p$ that a "bad" fruit will fall instead of the original fruit. The agent receives a higher reward for moving away from bad fruits, denoted by $|x_p - x_f|$. An additional high negative reward, -100, is given when $x_p = x_f$, because the space directly under a bad fruit represents a high-danger region. The agent does not have the fruit type feature in its representation, so it lacks the "true" representation of the real world. Without the ability to distinguish between fruit types, the agent can never learn the optimal policy.

The second domain is a variation of the game Flappy Bird. The goal is to fly a bird through the space between two pipes. The state in simulation is represented by $[y_t, y_b, y_a, v_a, \Delta x]$, where y_t, y_b, and y_a represent the y locations of the top pipe, bottom pipe, and agent, respectively; v_a is the agent's velocity; and Δx is the x distance between the agent and the pipe. The agent can either move up or take no action, in which case gravity begins to pull the bird downward. The simulation environment includes high pipes and low pipes, and the agent must learn to fly high above the ground and then swoop downward in order to pass between both low and high pipes. The agent receives +10 for getting past a pipe, -10 for crashing, and +0.1 any time it flies above a certain threshold (to encourage flying at a high altitude). In the real-world environment, pipes are composed of different materials (copper and steel), which the agent is unable to observe. Copper pipes close to the ground can cause a heavy wind to pass through, requiring the agent to be cautious and fly low, while steel pipes do not cause heavy winds, and thus the agent should fly at a high altitude before passing through them. The reward function remains the same for the real-world setting, except that the agent receives +0.1 for flying below a specific threshold (to encourage flying low) for copper pipes, and -100 when it flies high, because this represents a high-danger region. Without knowledge of the pipe's material, the agent is unable to learn the optimal policy for both pipes.

5.2 Oracle Simulation

We assume access to an oracle $O = \{A(s, a), \pi_{real}\}$ that provides feedback to the agent. We simulate this oracle by obtaining an optimal policy π_{real} for the target task and constructing an acceptable function $A(s, a)$ that specifies which actions are acceptable in each state. This function depends upon the domain, as well as how strict the given oracle is. A strict oracle may only consider optimal actions to be acceptable, while a lenient oracle may accept most actions except those that would lead to significantly lower Q-values.

To simulate different acceptable functions, we first trained an agent on the true real-world environment to obtain the optimal real-world Q-value function, Q_{real}. We then computed, for each state $s_{real} \in \mathcal{S}_{real}$, the difference in Q-values between the optimal action and every other action: $\Delta Q^i_{s_{real}} = Q_{real}(s_{real}, a^*) - Q_{real}(s_{real}, a_i), \forall a_i \in \mathcal{A}$. The set of all Q-value deltas, $\{\Delta Q^i_{s_{real}}\}$, quantifies all possible mistakes the agent could make.

The deltas are sorted in ascending order from least-dangerous to costliest mistakes, and the model chooses a cutoff delta value δ based on a specified percentile p at which

to separate acceptable and unacceptable actions. This cutoff value is used to define the acceptable function in an experimental setting – and, consequently, the set of blind spots (agent observations with at least one unacceptable action in a real-world state mapping to it) for the task. When $A(s, \hat{a})$ is queried with agent action $\hat{a}$, the oracle computes the difference: $\Delta Q_{\hat{a}} = Q_{real}(s, a^*) - Q_{real}(s, \hat{a})$. If $\Delta Q_{\hat{a}} < \delta$, action $\hat{a}$ is acceptable in state s; otherwise, the action is unacceptable. An acceptable function $A(s, a) = \{Q_{real}, p\}$ is defined by the target Q-value function Q_{real} and a percentile p for choosing the cutoff. A high p value simulates a lenient oracle, resulting in a greater number of acceptable actions and fewer blind spots. Consequently, more AM noise exists, because if the oracle accepts the majority of actions, there is a high chance that the agent will subsequently take an acceptable action that differs from the action of the oracle. A low p value simulates a strict oracle, because even actions with Q-values only slightly lower than the optimal will still be considered blind spots. This results in less AM noise, because deviation from the oracle is a strong indicator that the agent's action is truly unacceptable.

5.3 Baselines

The first baseline is a majority vote (MV) aggregation method for the noisy labels. For each state, MV takes the label that appears most frequently as the true label. The second baseline is all labels (AL), which uses no aggregation and simply passes all data points to a classifier. The model learning is the same for both our method and the baselines, which we used to assess the benefit gained from using DS for aggregation.

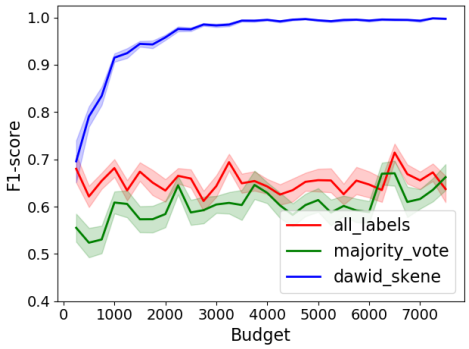
We report the baselines' performance when predicting blind spots based on the F1-score to assess both the precision and recall of the predictions, as well as the accuracy of the estimates of the likelihood of blind spots. We also compare results obtained with a strict versus a lenient oracle. The strict oracle was chosen such that only the optimal action was acceptable (no associated percentile p); for the lenient oracle, we used $p = 0.95$ for the Catcher domain and $p = 0.7$ for the Flappy Bird domain. We chose these percentile values to represent an acceptable function that was intuitive for these domains (e.g., unacceptable to be too close to a bad fruit). Varying p will affect the ground truth set of blind spots and in corrections, it will change when the oracle provides corrective feedback.

6. Results

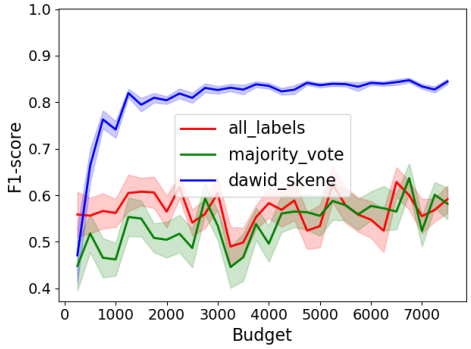
Here, we present the results of our approach in both domains. We observed better performance with our method compared with existing baselines, and also noted that different forms of feedback induced biases within the data that affected the learned blind spot model.

6.1 Benefits of Aggregation

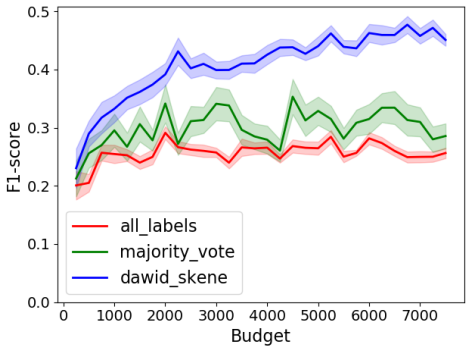
We first compared the performance of our approach, which uses DS for aggregation, to existing baselines: majority vote (MV) and all labels (AL), on the Catcher domain. In this section, we focus on feedback types with AM noise, because DS provides the greatest benefit when noise cannot be easily recovered by simple techniques. We present blind spot prediction performance for states visited during data collection, as this demonstrates the difference between our approach and the baselines with regard to the ability to estimate and



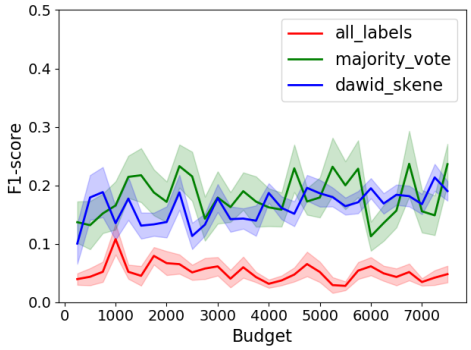
(a) R-AM with a strict oracle



(b) R-AM with a lenient oracle



(c) D-AM with a strict oracle



(d) D-AM with a lenient oracle

Figure 5: A comparison between our approach and baseline methods using random and demonstration data and varying oracles within the Catcher domain.

reduce noise in the training data. As depicted in Figure 5, we varied the number of oracle labels received by the agent (the budget) and reported the resulting F1-scores, which were weighted according to the "importance" of states (represented by how often the states were visited by π_{sim}). We ran the full experiment 25 times, and plotted the average performance along with standard error bars. We further compared all of the approaches against one another when applied with a strict versus a lenient oracle.

For the randomly sampled data, DS performed much better than MV and AL with both strict and lenient oracles due to the uniformity of labels across all observations. Because random samples are drawn from a uniform distribution, the agent receives observations about states that correspond with blind spots that are not all visited when actions are drawn according to the agent or oracle policy. DS could thus recover the prior and confusion matrix that generated this data, while MV predicted most regions to be safe because safe signals were much more common. AL exhibited lower accuracy because it did not aggregate the labels, which resulted in a poor prior estimate of blind spot regions.

Performance dropped overall with demonstration data (compared to random data), since feedback was biased based on the oracle's policy, preventing the system from learning about some blind spots that the agent would face during real-world task execution. Despite the decrease in performance, DS still performed well compared with both MV and AL in the presence of a strict oracle. With a lenient oracle, many safe regions were associated with unacceptable labels (due to action mismatch noise) that an unsupervised learning method such as DS could not completely recover. Nevertheless, DS performed equally well to MV, and much better than AL.

Overall, DS performed well compared with the baselines; however, performance dropped when states were sampled in a biased form rather than randomly, and also with a lenient oracle rather than a strict one. We observed similar trends in the Flappy Bird domain with regard to the benefit of DS versus baselines. We discuss details of the effect of feedback types and resulting biases across the two domains in Section 6.3.

6.2 Effect of Feedback Type on Classifier Performance

Next, we evaluated the best-performing approach (learning with DS) while varying the oracle feedback type. We evaluated the classifier on states observed in oracle feedback, which measures the ability of DS to recover true labels from noisy ones; and on unseen data, which highlights the ability of the classifier to generalize to unvisited regions. We report F1-scores in Table 2 for each condition.

The results show that learning from random data performed well across conditions, regardless of whether the condition included SR noise (R-A) alone or both SR and AM noises (R-AM). R-AM performed well in these cases, despite the presence of AM noise, because DS was able to recover the labels for randomly sampled data when the labels were distributed uniformly across all agent observations. However, the performances of both R-A and R-AM dropped when the oracle was lenient, as there were fewer blind spots to learn from.

Our findings also indicate that the correlated nature of observations from demonstrations and corrections overall reduced the performance of classifiers compared with random data feedback types. In the strict oracle case, the performances of D-A, D-AM, and C

Table 2: The effect of feedback type on classifier performance in the Catcher domain, reported as F1-scores.

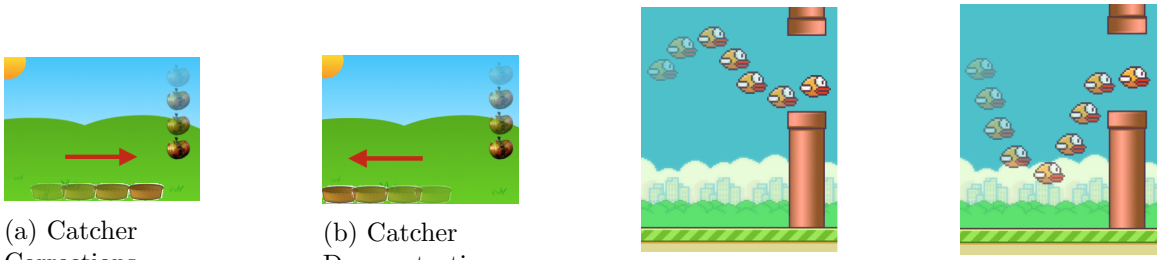
	Strict		**Lenient**	
Feedback Type	Seen	Unseen	Seen	Unseen
R-A	0.996	0.994	0.825	0.700
R-AM	0.997	0.993	0.837	0.833
D-A	0.476	0.453	0.084	0.152
D-AM	0.487	0.477	0.209	0.274
C	0.478	0.461	0.636	0.520

were comparable because AM noise was low, resulting in similar feedback obtained from corrections and demonstrations, because the monitoring oracle that corrected the agent would redirect the agent any time it deviated from the optimal. On the other hand, a lenient oracle would only correct an agent if an action would be very dangerous, resulting in a more informative state distribution than that obtained through demonstrations. In this case, both versions of demonstrations failed to collect observations about major blind spot regions. Furthermore, D-A yielded few unacceptable labels due to the absence of AM noise; this hurt the prior estimate and made it difficult for the system to learn an accurate classifier. Thus, we report that D-A's performance was even worse than that of D-AM for this scenario.

6.3 Effect of Feedback Type on Oracle-in-the-Loop Evaluation

The ultimate goal of our work is to use limited feedback to learn a blind spot model of the real world that could enable an agent to act more intelligently. Ideally, the agent could use this model to selectively query for human help, avoiding costly mistakes without overburdening the human. The next set of results evaluates the effectiveness of the learned model in oracle-in-the-loop (OIL) execution. In this case, the agent executes actions in a real-world environment using the source policy. When the learned model predicts a state to be a blind spot using the learned calibrated threshold, the agent queries an oracle for the optimal action, then takes this provided action and resumes acting according to its policy in the world. We compared our method of querying the oracle based on the learned model to an insufficiently cautious agent that never queried and an overly conservative agent that always queried.

Figure 6 shows that in both domains, demonstrations and corrections provided different feedback, and thus introduced separate biases into the data. Corrections provided direct feedback about the actions the agent would take, while demonstrations followed the policy of an optimal oracle. In the Catcher domain, an optimal oracle moved toward good fruits and away from bad fruits; an agent trained in the source task with only good fruits learned to move closer to all fruits. In Figures 6a and 6b, the fruits represent the movement of a bad fruit, and the agent moving at the bottom represents the feedback bias. Demonstrations provided observations about states that were far away from bad fruits, while corrections provided observations about states closer to bad fruits. In the Flappy Bird domain, the agent was required to be careful and fly low around copper pipes; as depicted in Figures



(a) Catcher Corrections

(b) Catcher Demonstrations

(c) Flappy Corrections

(d) Flappy Demonstrations

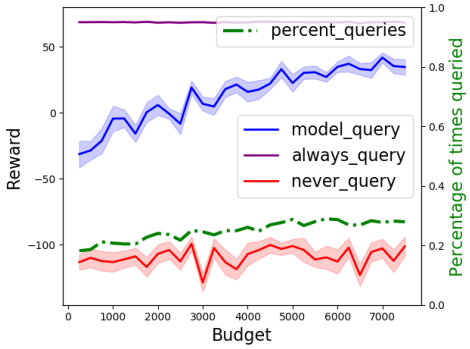
Figure 6: The data bias of demonstrations and corrections observed in both the Catcher and Flappy Bird domains.

6c and 6d, a demonstration would show the agent flying low for a copper pipe, while a correction trajectory would allow the agent to fly slightly higher and correct only before it went too far. This provides information about the more informative states the agent would likely visit.

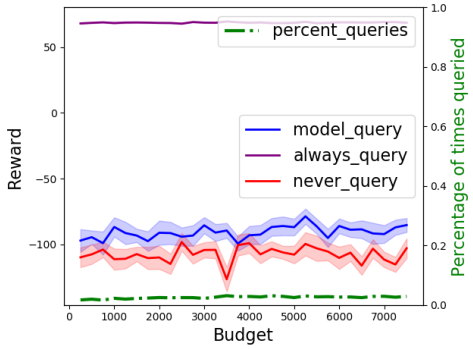
Figure 7 depicts the performance of our model on OIL evaluation with different feedback types. We report performance as the number of oracle labels (budget) increases. The left y-axis shows the reward obtained in the real-world environment by an agent that used the model to query compared with an agent that never queried (NQ) and one that always queried (AQ). On the same graph, the dotted line indicates the percentage of times the agent queried the oracle for help using our model. Across all feedback types, the model achieved higher reward than the NQ agent, while still querying with relative infrequence.

Figures 7a and 7c show that, with a lenient oracle, D-AM and C both yielded greater rewards than NQ, while C had a lower percentage of queries. C, however, did obtain less reward than D-AM, because corrections did not receive labels about many blind spot regions, as the oracle steered the agent away from dangerous areas before the agent actually visited them. Thus, when the agent began the task close to a blind spot region, the model was uninformed about that part of the world, and thus the agent did not know to move away. In Section 8.1, we introduce a modification to our approach that allows the agent to learn about the blind spot regions it did not visit by estimating a dynamics model of the world and propagating oracle labels to potential future states.

Interestingly, even though D-AM did not earn a high F1-score on classifier performance, it performed well during OIL evaluation because D-AM considered any action mismatch (where the agent deviated from the optimal action) as an unacceptable label, resulting in an overly conservative agent that queried for help at all mismatches. For example, in the Catcher domain, when the oracle was far away from a bad fruit and moving further from it, D-AM marked these regions as blind spots due to action mismatches. When the agent queried for help in these states, the oracle instructed the agent to move away. For D-A (Figure 7b), states far from the fruit were resolved as safe; the agent thus moved towards the fruit, but because there was never any demonstration data in which the agent was close to bad fruits, D-A did not know to act. With a strict oracle (Figure 7d), all regions with action mismatches were considered blind spots, so D-A did not have this issue and queried the oracle in the same states as D-AM.



(a) D-AM with a lenient oracle



(b) D-A with a lenient oracle

(c) C with a lenient oracle

(d) D-A with a strict oracle

Figure 7: Oracle-in-the-loop evaluation in the Catcher domain with varying feedback types.

Table 3: Reward and percentage of times queried for oracle-in-the-loop evaluation in the Flappy Bird domain.

	Strict		**Lenient**	
Feedback	Reward	% Queries	Reward	% Queries
AQ	12.69	100%	12.66	100%
NQ	-318.77	0%	-316.17	0%
D-A	-151.15	25%	-373.46	5%
D-AM	-197.84	20%	-186.46	23%
C	-147.21	21%	-125.44	14%

We observed similar trends in the Flappy Bird domain, as shown in Table 3. With a strict oracle, all three feedback types performed similarly and better than NQ, while also querying much less frequently than AQ. With a lenient oracle, D-A performed poorly because the initial states (where the oracle demonstrated how to fly low and between the pipes) were resolved to be safe. The agent thus did not query and ended up flying high, resulting in a substantial negative reward. Note that the classifiers used during OIL evaluation valued precision as much as recall, meaning that they put equal emphasis on making a mistake and querying the oracle unnecessarily. In cases where errors were more costly than querying, the classifier threshold could be chosen accordingly to increase the aggregate reward of oracle-in-the-loop execution in exchange for a larger percentage of queries to the oracle.

The key takeaway is that the type of feedback to use depends highly on the domain. Corrections provides the most informative signals on the agent's policy but can be more expensive as it requires a human to be constantly monitoring a system. Further, in real-world applications, human corrective feedback will likely be delayed, requiring an agent to associate the feedback with some prior action, introducing another research challenge. Demonstration data is often easier to obtain, but it provides feedback in a different region of the world (based on the oracle's policy). This can either result in a very conservative agent that avoids dangerous regions through frequent querying or an agent that navigates into uncharted territory and makes costly errors because the blind spot model is no longer accurate. Depending on the constraints of a particular problem, the most suitable feedback type for accurate blind spot learning may vary.

7. Relaxing Optimality Assumptions Analysis

Here, we present analyses of our approach when we relaxed two of our initial optimality assumptions. The first assumption we made in our formulation was that the oracle policy π_{real} was optimal in the real world, resulting in optimal demonstrations. In this section, we evaluate the performance of our approach when the oracle's policy was suboptimal in the real world. The second assumption we made was that the policy π_{sim} learned from simulation was optimal; we also present an analysis of our approach with a suboptimal π_{sim}. Finally, we evaluate our method when we relaxed both assumptions simultaneously.

7.1 Suboptimal Oracle Policy π_{real}

In previous results, we assumed that the agent received demonstrations from an oracle that always followed the optimal policy when acting in the real world. We relaxed this assumption and incorporated an oracle that selected acceptable, but not necessarily optimal, actions. To simulate a suboptimal oracle, in each state $s_i \in \mathcal{S}_{real}$, a random acceptable action a_i from the set of all possible acceptable actions $\{a \in \mathcal{A} | A(s_i, a) = 0\}$ was chosen by the oracle, where $\mathcal{A}$ represents the set of all possible actions and $A(s_i, a)$ is the acceptable function. Collecting demonstrations from a suboptimal but acceptable oracle increases the variety of acceptable and unacceptable labels in demonstration data, which can provide more information to the agent. However, this also leads to a larger number of noisy signals for blind spot discovery, because mismatches become more prevalent throughout.

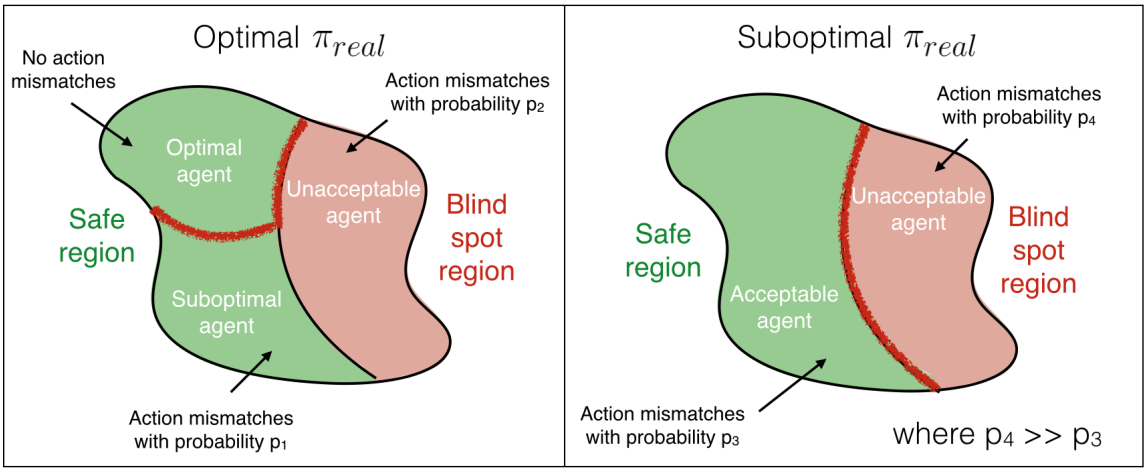


Figure 8: A visualization of differences in the data when an oracle followed an optimal vs. a suboptimal policy. The red lines represent the separation boundaries that the Dawid-Skene algorithm learned in order to distinguish safe and blind spot regions.

7.1.1 Difference in Data

First, we discuss the differences between the labels the agent receives when an oracle follows an optimal policy vs. a suboptimal but acceptable policy. We analyze the data by observing the ratio of acceptable and unacceptable labels for safe and blind spot regions.

In an ideal scenario, for DS to properly distinguish between the two different categories, we want the proportion of acceptable and unacceptable labels to appear consistently different between safe and blind spot regions. As depicted in Figure 8, we observed that with an optimal oracle providing demonstrations, two clusters within the safe region received different proportions of labels. One cluster of the safe region had no action mismatches, because these were areas that appeared in the simulation task with good fruits only, in which the agent and oracle always acted optimally. Another cluster in the safe region existed, which represented areas in the real world in which the agent acted suboptimally, but still acceptably; thus, both action matches and mismatches existed. Given the presence of two different clusters in the safe region, DS, an unsupervised learning method, learned to separate the safe regions with no action mismatches from all other regions; the red line in Figure 8 conceptually represents what DS learned.

To better understand the two safe clusters, consider the Catcher example: only good fruits fall on the left-hand side of the game, which is identical to the simulation environment. If the oracle acts optimally here, its actions and those of the agent will always match, and the agent will never receive an unacceptable label in this safe region. On the right-hand side of the game, good fruits fall with a certain probability; otherwise, bad fruits fall. Some parts of this region are safe, specifically areas where the agent might perform a suboptimal but acceptable action that differs from the oracle's optimal action, resulting in action mismatches.

When an oracle follows a suboptimal policy, there are no longer two separate clusters in the safe region. When the real world matches the simulation, the agent's optimal action

π_{real}, π_{sim}	DS Accuracy	F1 Seen	F1 Unseen	OIL Reward	OIL % Queries
Opt, Opt	0.477	0.134	0.258	44.855	0.288
Sub, Opt	0.929	0.628	0.396	-9.721	0.154
Opt, Sub	0.388	0.333	0.332	53.862	0.512
Sub, Sub	0.944	0.481	0.295	17.694	0.090

Table 4: A comparison of aggregation, classifier, and oracle-in-the-loop performance in the Catcher domain when π_{sim} and π_{real} were each set to both optimal and suboptimal.

and the oracle's suboptimal action no longer always match. Figure 8 shows that only two regions exist with a suboptimal oracle, because all agent observations receive both acceptable and unacceptable labels from action mismatches. Interestingly, even though the number of mismatches increases overall, the labels in the safe region become more homogeneous, making it easier for DS to separate the two categories.

7.1.2 Aggregation and Classifier Performance

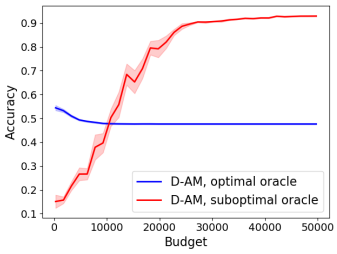
Figure 9a depicts the accuracy of the DS algorithm when obtaining data from an optimal vs. a suboptimal oracle. The graph shows that with a low budget, DS' accuracy is lower with a suboptimal oracle policy π_{real}; this is because the proportions p_3 and p_4, as shown in Figure 8, are not easily distinguishable. As the budget increases, the data from a suboptimal policy helps DS to achieve greater accuracy because the ratio of acceptable and unacceptable labels begins to differ between the safe and blind spot regions. Specifically, the percentage of unacceptable labels from action mismatches is much higher in blind spot than in safe regions (i.e., $p_4 \gg p_3$). With more action mismatches, it would intuitively seem that differentiating between categories would become more difficult; however, with sufficient data, the action mismatches are helpful for separating the two groups more easily.

The first two rows of Table 4 show the performance of each part of our pipeline when the optimality of the oracle policy π_{real} was relaxed. The table depicts DS' accuracy when predicting safe and blind spot regions, the performance of the classifier on seen and unseen data, and oracle-in-the-loop performance based on the predicted blind spots. (All of these results are based on a high budget of 50,000 labels from the oracle.) DS' accuracy improved when π_{real} changed from optimal to suboptimal, which is consistent with Figure 9a.

For classifier performance, having a suboptimal oracle resulted in better F1-scores than using an optimal oracle. For seen data, F1-scores were predominantly based on DS' accuracy, so it follows that the classifier predicted seen data well when DS had high accuracy. The performance on unseen data was lower overall than seen data because it is difficult to generalize well, but the suboptimal oracle condition still led to better performance than the optimal condition.

7.1.3 Oracle-in-the-Loop Evaluation

We next analyze whether high aggregation performance translates to high performance during oracle-in-the-loop (OIL) evaluation with suboptimal demonstration data. Figure 9b indicates that with sufficient budget, an agent trained on optimal demonstrations could get relatively close to achieving the optimal reward. Figure 9c shows that OIL performance



(a) The accuracy of the Dawid-Skene algorithm when estimating safe and blind spot regions given data from an optimal vs. a suboptimal oracle.

(b) Optimal π_{real}, Optimal π_{sim}. The agent could effectively avoid dangerous blind spot regions and perform well in OIL evaluation with a sufficient budget.

(c) Suboptimal π_{real}, Optimal π_{sim}. The agent performed worse with a larger budget, because the prior of blind spots was estimated to be lower.

Figure 9: A comparison of an optimal vs. a suboptimal oracle policy π_{real} within the Catcher domain. Figure 9a depicts DS' accuracy, and Figures 9b and 9c show OIL performance.

actually dropped with suboptimal demonstrations as budget increased; this is because even though Dawid-Skene's performance was high, the prior of blind spots was predicted to be lower than the true ground truth prior. With a larger budget, DS was more certain about its predictions, but estimated a lower prior of blind spots. Our current approach chooses to query in the world based on a threshold selected during model learning, which aims to predict a similar percentage of blind spots in the calibration data as the training data. As the prior was estimated to be lower than the ground truth for the suboptimal oracle, the threshold was selected to be very high in order to prefer fewer predicted blind spots.

To evaluate the performance of our model invariant of the threshold, we compared the ROC curves between the optimal and suboptimal oracle. Figure 10a indicates that with an optimal oracle, the learned classifier did not predict much better than a random classifier, with an AUC of 0.6. On the other hand, a classifier trained on suboptimal data yielded an AUC of 0.96 (as shown in Figure 10b) because it was able to separate safe and blind spot regions well across different settings of thresholds.

In order to understand the effect of prior estimation on OIL execution, we compared the performance of an optimal and a suboptimal π_{real} when we fixed the prior to be the true prior of blind spots in the data. This is, of course, an unrealistic condition, because the agent would never know the true prior of blind spots; however, it shows that the model can learn blind spots well if the prior can be better estimated. Figure 10c shows that with a fixed prior of blind spots, an agent learning from suboptimal data can achieve much greater reward than an agent learning from optimal data with the same prior.

Since we cannot set this manually to an unknown prior in the real world, we can instead combine the strengths of both optimal and suboptimal data. Optimal demonstration data helps the model to learn a more conservative prior, since many safe regions are predicted to be blind spots; suboptimal demonstrations provide more uniformly distributed labels that make it easier to disambiguate safe and blind spot groups. If there was an option to query the oracle for specific forms of data, the agent could first query for a small amount

of optimal data to learn a prior, and then ask for more data from a suboptimal policy that would help to better predict blind spots in seen data. Figure 10d indicates that an agent could indeed perform better by combining both forms of data. In our experiment, given a particular budget on the x-axis, the agent queried for optimal data with 33% of its budget and used these labels to estimate the prior of blind spots. It then queried for suboptimal data with 67% of its budget to perform aggregation and predict blind spot probabilities for each agent observation. The learned classifier used the prior estimated from optimal data to set the threshold, resulting in much better performance during OIL evaluation than with suboptimal data alone.

Overall, by using suboptimal instead of optimal demonstration data, the agent was able to better predict safe and blind spot regions. In terms of real-world performance, the agent that learns from optimal demonstrations earns a greater reward because it is more conservative. This is an important point, because it demonstrates that classifier performance on the test data is not the only measure to evaluate. Even with poor performance of blind spot prediction, an agent could avoid blind spot regions altogether by being slightly more conservative. In order to achieve both high aggregation and high OIL performance, one approach is to learn the prior with a small amount of optimal data, then learn the blind spot classifier with a set of suboptimal demonstrations; thus, the agent could benefit from both forms of data for better overall performance.

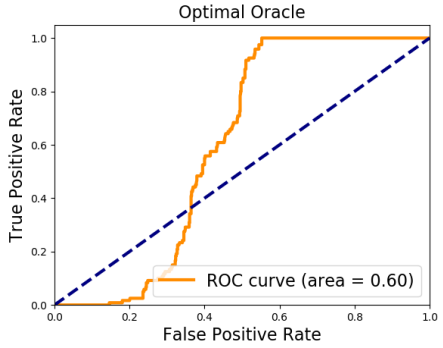
7.2 Suboptimal Simulator Policy π_{sim}

Another assumption we made was that the agent's policy π_{sim} learned from simulation would be optimal with respect to the simulation world. Learning an optimal simulation policy may not always be possible if the agent has limited training time; thus, we observed the effect that a suboptimal policy has on the agent's ability to identify blind spots and perform well in the real world with an oracle in-the-loop. The agent trained in the simulation environment for a limited time, and we evaluated our approach with this suboptimal π_{sim}.

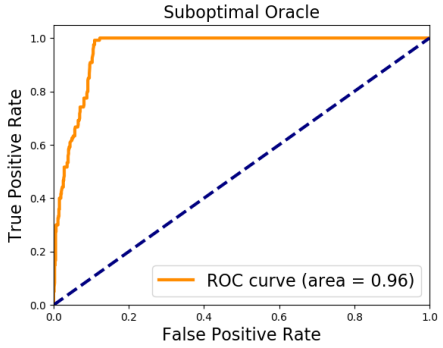
7.2.1 Difference in Data

In terms of differences in the collected data, we observed a greater number of unacceptable labels from action mismatches for both safe and blind spot regions using a suboptimal simulator policy, as compared to an optimal one. This is to be expected, because given that the agent is taking suboptimal actions, there is a greater likelihood that the agent's action will not match that of the oracle in safe regions. Similarly, in blind spot regions, the agent's action will match the oracle's less frequently; for example, in a blind spot region where a bad fruit is close to the agent, two potential corresponding real-world states exist: one in which a bad fruit is close, and one in which a good fruit is close. If the agent's action is suboptimal in the "good fruit" region, there will be more action mismatches, even for blind spot regions.

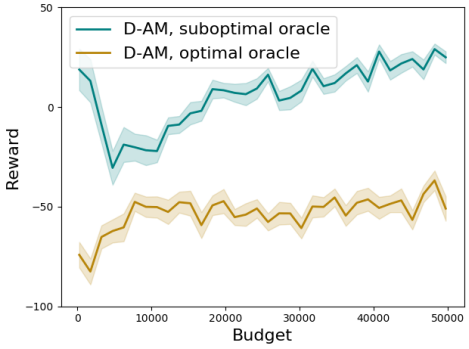
Taking a closer look at the label distribution, when π_{sim} is suboptimal, many safe regions never receive a "safe" label. The agent always takes the suboptimal action in these states, while the oracle always takes the optimal action. Thus, the agent will only receive action mismatch labels, making it impossible for DS to predict these agent observations as safe.



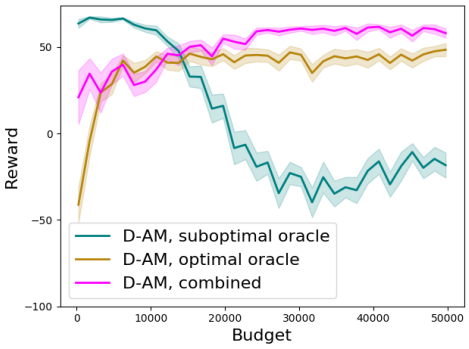
(a) ROC curve for an optimal π_{real} with an AUC of 0.6.



(b) ROC curve for a suboptimal π_{real} with an AUC of 0.96.



(c) Optimal vs. suboptimal oracle with a fixed prior.



(d) Combined optimal+suboptimal on OIL evaluation.

Figure 10: A comparison of optimal and suboptimal π_{real} demonstrations in the Catcher domain through an analysis of the ROC curve. A classifier trained with suboptimal data (Figure 10b) was better able to separate safe and blind spot regions compared with one trained with optimal demonstration data (Figure 10a).

(a) Optimal π_{real}, Suboptimal π_{sim}. The estimated prior of blind spots from DS was high, so the agent queried frequently and performed well.

(b) Suboptimal π_{real}, Suboptimal π_{sim}. The agent's OIL performance decreased as budget increased.

Figure 11: OIL performance with a suboptimal π_{sim} in the Catcher domain. Performance is compared between an optimal and suboptimal π_{real}.

7.2.2 Aggregation and Classifier Performance

When the agent learns an optimal simulator policy, the Dawid-Skene algorithm learns to separate observations with only "safe" labels from everything else. With a suboptimal simulator policy, there is an increase in the number of agent observations with only action mismatches. Thus, DS learns to separate regions with action mismatches alone from all other observations, resulting in similar accuracy for both optimal and suboptimal policies on aggregation. However, the suboptimal π_{sim} obtains a better F1-score than the optimal π_{sim}, because the prior from DS was estimated to be high for blind spots. The threshold chosen during model learning allows for prediction of a greater number of blind spots.

7.2.3 Oracle-in-the-Loop Evaluation

Because DS predicts a higher prior for blind spots, a suboptimal π_{sim} actually performs better than an optimal π_{sim} during OIL evaluation. The agent predicts many safe regions to be blind spots and queries the oracle for help; this allows the agent to avoid dangerous regions, but also results in more unnecessary (and undesirable) querying.

Also, the worst possible reward that an agent can receive by never querying is higher for a suboptimal simulator policy compared with an optimal simulator policy. This is not intuitive, but is an artifact of our domain and reward function. In the Catcher domain, if the agent consistently moves toward a bad fruit, the agent receives a negative reward. When the agent takes a suboptimal action, it actually receives a better reward: for example, if a bad fruit is present, the agent interprets it as a good fruit and would move closer if π_{sim} were optimal, resulting in some reward, x. However, if the agent were to act suboptimally in that state, it would either remain stationary or move left, both of which would result in $> x$ reward.

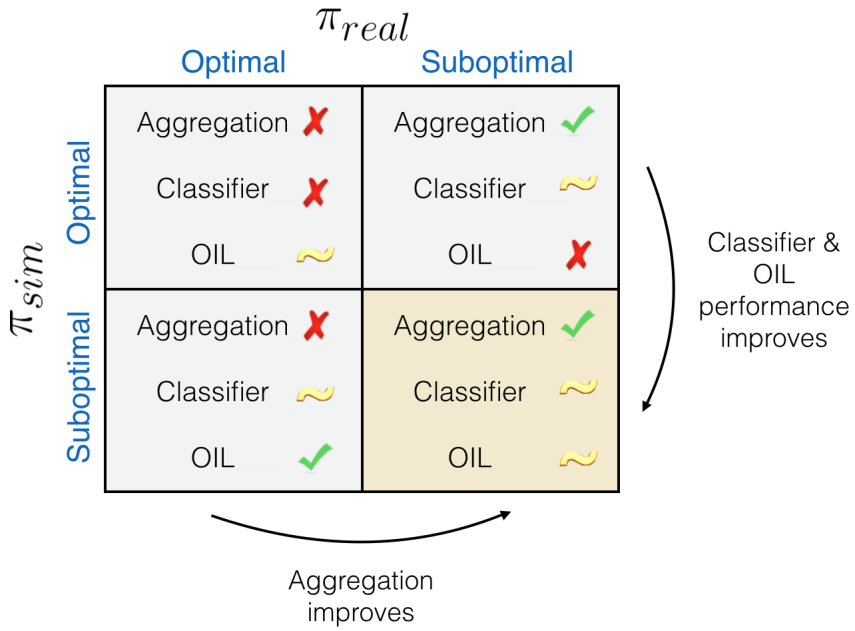


Figure 12: This figure summarizes the results from relaxing π_{real} and π_{sim}. When π_{real} was suboptimal, it became easier for Dawid-Skene to aggregate labels. When π_{sim} was suboptimal, the prior of blind spots increased, improving classifier and oracle-in-the-loop performance.

7.3 Suboptimal Oracle Policy π_{real} and Suboptimal Simulator Policy π_{sim}

When both assumptions were relaxed, performance improved over that observed when either assumption was relaxed separately. Figure 12 summarizes the results of each possible variation of π_{real} and π_{sim} being optimal vs. suboptimal. The horizontal axis varies π_{real}, and the vertical axis varies π_{sim}. The figure highlights changes to aggregation, classifier, and OIL performance when moving from one assumption box to another.

By receiving more varied demonstrations from a suboptimal π_{real}, the agent can better separate safe and blind spot regions because the proportions of acceptable and unacceptable labels are more separable between the two classes, resulting in improved aggregation performance. With a suboptimal π_{sim}, the prior of blind spots predicted from DS increases, resulting in a threshold that predicts a greater number of states as blind spots. This leads to a conservative agent that queries much more frequently, thus achieving better reward on the task.

8. Data Augmentation Improvements

We now present two improvements to our pipeline that boost performance when working with correction and demonstration data. The first improves the performance of corrections: estimating the transition model of the world and propagating unacceptable signals to later states even when the agent never visits those states (as they are dangerous and the oracle will not allow the agent to go there). The second improves demonstrations: augmenting

(a) Original corrections data. The agent was unable to achieve the same performance because it lacked signals about blind spot regions that it never visited.

(b) π_{sim} in the real world, without action corrections. (Note this is an unsafe and infeasible data collection method, and is only used as a comparison to corrections.)

(c) Corrections along with additional labels propagated by estimating the transition model of the real world. The agent achieved performance similar to π_{sim} without corrections, but *safely*.

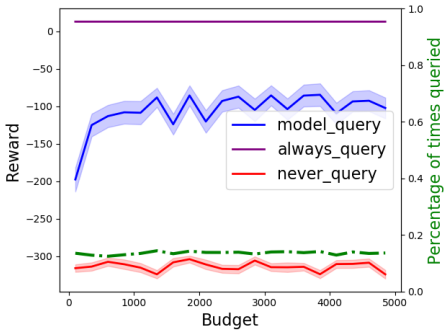
Figure 13: The system's performance in the Catcher domain when propagating unacceptable labels to future states based on an estimated transition model of the world. This resulted in improved performance with corrections data, while also maintaining safety.

demonstration data with corrections to have better exposure in the real world, and thus improved prediction of blind spots.

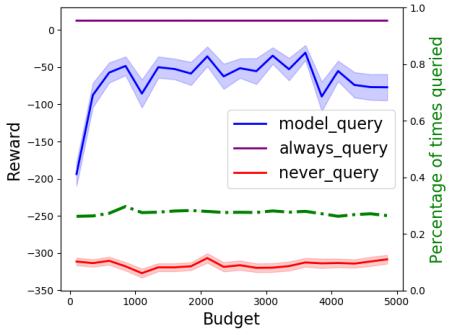
8.1 Boosting Corrections Performance

The first improvement is intended to increase performance when using corrections data. When the agent obtains labels from corrections, it receives direct signals about its own actions. Consequently, the unacceptable labels the agent receives are true labels, not based on action mismatches. The issue, however, is that the agent never receives labels in dangerous blind spot regions, since the oracle always steers the agent away before the agent reaches that area. Figure 13a depicts the performance of an agent using corrections to predict blind spots within the Catcher domain: it was unable to achieve high OIL performance because when it began the task close to blind spot regions, it did not have a good model of blind spots and therefore had not learned to avoid them.

To analyze the impact of an agent not receiving signals when close to blind spot regions, we ran an experiment with a modification in the data collection for corrections. With the original corrections feedback type, the agent receives labels from the oracle, and in the event of unacceptable labels, the oracle steers the agent away from blind spots by providing the optimal action to take at that state. With our modification, the agent simply uses π_{sim} directly in the real world during data collection, and obtains acceptable and unacceptable labels without being corrected. This allows the agent to receive true labels for many more blind spots, resulting in more accurate blind spot modeling. However, running π_{sim} without corrections cannot be safely transferred to the real world; we include the analysis here only in order to assess the benefit of the agent receiving signals about blind spot regions. Figure 13b indicates that this modified form of corrections feedback performed much better than the original during OIL evaluation in Catcher.



(a) This approach uses original corrections data to learn a blind spot model.



(b) This approach uses corrections with propagated labels based on potential future states the agent might visit.

Figure 14: Performance in the Flappy Bird domain with the corrections augmentation.

Since running π_{sim} without action corrections can be dangerous and may involve costly errors, the agent can instead propagate the unacceptable labels it receives to future states based on which regions the agent thinks it will go to. In other words, the agent can safely obtain data using the original corrections feedback type (in which the oracle steers the agent away from dangerous regions). However, to better model blind spot regions without labels from the oracle, the system can estimate the dynamics model of the world and propagate unacceptable labels to potential future states. To do this, the system first collects data from corrections in the original form (with action corrections), providing the agent with acceptable and unacceptable signals. Our system then estimates the transition model of the real world, $\hat{T}_{real}(o_{agent}, a, o'_{agent})$, using this corrections data. This transition model estimate is based on the agent observation space, $o_{agent} \in \mathcal{S}_{agent}$, because this is the representation the agent has when recording data from the oracle. We chose to estimate the transition model of the real world, rather than the one in simulation, because this is more representative of the world that the agent will operate in. If the agent has access to very little real-world data, however, another possible approach would be to estimate an approximate transition model of the simulation world and use it as a proxy for estimating real-world dynamics.

Given $\hat{T}_{real}(o_{agent}, a, o'_{agent})$ and the labels obtained through corrections, our system propagates every unacceptable label a few steps forward. This can then be modified to propagate a decaying unacceptable label over a longer sequence. The aim of propagation is to provide better labels for blind spot regions that the agent never visited. However, by artificially creating these new labels, our system may unintentionally apply unacceptable labels to safe regions, which could result in overprediction of blind spots. However, as long as overprediction is minimal, this method remains preferable to the original corrections method because it results in a slightly more conservative and safer agent. Figure 13c indicates that when the system estimates the transition model and propagates unacceptable signals, the system obtains similar performance to π_{sim} on the Catcher domain, while also maintaining safety.

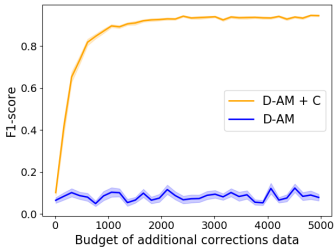
This improvement for corrections data leads to higher performance in the Flappy Bird domain as well. Figure 14a depicts OIL performance with the original corrections data: in this case, the agent was able to obtain much higher reward compared with an agent that never queried, and it only queried approximately 15% of the time. However, the agent was unable to obtain a maximum reward close to that obtained by an agent that always queried. In Figure 14b, we include the propagation of correction labels to potential future states; this addition improved performance during OIL evaluation to be closer to the best possible reward.

8.2 Boosting Demonstrations Performance

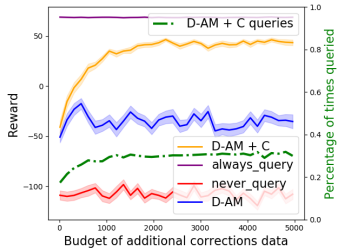
The second improvement involves boosting performance when using demonstrations, since demonstration data suffers from limited state exposure. In the Catcher example, demonstration data only includes states located near good fruits and far away from bad fruits. The agent never sees regions close to bad fruits; however, these are still important areas that the agent would visit in the real world. Earlier results indicated that an agent using demonstration data obtained low F1-scores on classifier performance, but was able to perform relatively well on OIL performance with sufficient budget by being extra-conservative.

To improve F1-scores and OIL performance, we analyzed a feedback type where the agent was able to review all action mismatches in demonstration data with the oracle (D-A). Our findings showed that reviewing these action mismatches did not improve performance – and, in fact, resulted in a performance drop. Regions with mismatches were resolved to be safe, leading the agent to unsafe regions for which it had no data and thus did not know to query the oracle. Since reviewing action mismatches in demonstration data did not help, another option would be to augment the space of visited states in demonstrations with a new set of states obtained from corrections data. Imagine that the agent has access to a set of demonstrations: to improve performance, the system could ask the oracle for a limited monitoring session in the real world in order to collect more data.

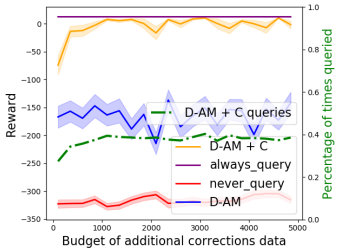
In our experiments, we set a constant budget of 250 labels obtained from demonstration data to which the agent already had access. We then observed the performance of our approach as demonstration data was increasingly combined with more data obtained through corrections. Figure 15a indicates that in the Catcher domain, the F1-score for seen data improved considerably with the addition of corrections data. Corrections data did not contain any action mismatches, and was therefore easier for the Dawid-Skene algorithm to separate. The performance improvement resulting from DS in turn improved classifier performance on both seen and unseen data. Figure 15b depicts OIL performance in Catcher of an agent that learned only from demonstrations (D-AM) vs. an agent that learned from both demonstrations and corrections data (D-AM + C). We used the modification proposed in Section 8.1 to propagate unacceptable labels for the corrections data, and found that D-AM + C performed considerably better than D-AM alone. In the Flappy Bird domain, we observed similar benefits from augmenting demonstrations data with corrections data. While keeping the amount of demonstration data constant at 1,000 labels, we increased the amount of combined corrections data; Figure 15c indicates the benefit to reward obtained in the real-world environment with this augmented data in Flappy Bird.



(a) Classifier performance on seen data when incorporating data from demonstrations alone vs. demonstrations augmented with corrections data in the Catcher domain.



(b) OIL performance when using data from demonstrations alone vs. demonstrations with additional corrections data in Catcher.



(c) OIL performance in Flappy Bird comparing demonstrations alone with demonstrations and additional corrections data.

Figure 15: Assessing the benefit of combining demonstrations data with additional corrections data to increase exposure and improve performance.

9. Discussion

In this work, we formulate the problem of identifying blind spots of AI systems that occur due to limited state representation. Many current safe AI methods assume sufficient representation for learning and acting (Garcıa & Fernández, 2015; Saunders et al., 2017; Kahn et al., 2017), but when that representation is insufficient, these approaches can break down. Our system explicitly learns how to find these errors through the help of an external oracle or human; using this feedback, agents can better understand their own failures and query for help accordingly.

We envision a future with safer deployment of more self-aware AI systems, allowing for iterative refinement with safely obtained real-world data. Our approach to learning blind spots provides a first step toward using human feedback to identify agent failures. Given an agent's flawed representation, we introduce a step to intelligently aggregate signals provided by an oracle operating within a different representation, and demonstrate in our experiments that our blind spot model enables an agent to avoid costly mistakes while querying with relative infrequence. This approach identifies blind spots due to representation incompleteness. Other types of blind spots, such as those due to incorrect models of reward or transitions, can also be included into the approach to enable an agent to better understand its errors.

There are many interesting steps for future work. Our current approach still requires sufficient oracle data to learn blind spots. Active learning can be used to reduce the number of labeled data points by intelligently querying for labels at important states. One challenge in this setting is that the agent must query for trajectories rather than single data points (as in supervised learning). This requires the agent to reason about possible futures and select a trajectory query that will lead to the greatest information gain across the full set of points. To enable the human to provide richer feedback, the model can also be relaxed to take a continuous quality rating from the human, rather than a binary label for acceptable/unacceptable actions.

Generalizing human feedback to communicate information not only about actions but also about states can be challenging because the representations of the agent and the human do not match. One solution is to query through an interpretable communication channel that leverages an easy-to-understand structure. One such example would be to use linear temporal logic (Shah, Kamath, Shah, & Li, 2018a) or object representations (Diuk, Cohen, & Littman, 2008; Ramakrishnan, Narasimhan, & Shah, 2016) as a shared medium or to use other forms of data querying, such as preference elicitation (Christiano et al., 2017; Bıyık & Sadigh, 2018). To reduce the burden on the human helper, the agent could also make a more informed decision about whether to query or proceed with an action if the agent knew the relative cost of making a mistake in the world vs. the cost of querying an oracle. Adding such decision-theoretic analyses can be a principled way of deciding how to act in the real world using the learned blind spot model and can improve human-agent coordination.

In future work, we would like to apply our method to more complex problems and richer real-world applications. Specifically, some avenues for future research include evaluating with real human users, scaling up the approach to high-dimensional state representations, and applying the method to specific applications (such as autonomous driving and robotics). Collecting data from human users can introduce new challenges, such as access to even smaller sample sizes and the presence of additional forms of noise. As discussed earlier, active querying to users represents one way of learning good models when the cost of obtaining data is high. Human feedback in real-world applications can have added noise due to the delay in correction signals. Realistically, people cannot provide action corrections instantaneously due to slow human reflexes, which can cause an agent to misinterpret signals. Thus, it is important for the agent to learn how to perform credit assignment and penalize the most likely action the human was referring to. Human studies could help us better understand how to augment our blind spot model learning.

In addition to working with people, we would like to explore how our approach could be modified to work in high-dimensional and continuous state representations. First, an agent's policy learned from simulation will likely be a high-dimensional neural network. Similarly, the oracle's acceptable function and policy will also be high-dimensional. In the data collection step, a human would provide labels at states, but because states can be continuous, many states will not map to a single agent observation. To address this, we can introduce an additional clustering step to identify similar states or learn a set of domain-agnostic features (Stadie, Abbeel, & Sutskever, 2017) that capture the essential components of the task independent of visual appearance. Representing states in terms of a domain-agnostic, lower-dimensional space can also make learning more tractable in continuous settings. For example, estimating a transition model of the world to propagate human signals to future dangerous states can improve our prediction of blind spots (Section 8.1), but it may be difficult to estimate in a high-dimensional space. Thus, we can instead estimate a lower-dimensional transition model that is more tractable.

Once these labels are assigned to similar states in a lower-dimensional space, we can use our aggregation step to combine these labels. The model learning step may involve building a larger and more complex model, like a deep neural network, that can capture rich relationships. When data is unbalanced in continuous problems and oversampling and undersampling do not provide enough diversity, data augmentation techniques can be used. Training a blind spot model based on domain-agnostic features can lead to more robust

predictions in the real world. Finally, using our approach in real-world applications would be an interesting and impactful direction for future work. Understanding the types of blind spots that exist in today's AI systems (such as self-driving cars and commercial robots) could inform the addition of other improvements to our method.

10. Conclusion

In this work, we addressed the challenge of discovering agent blind spots in reinforcement learning when the state representation of an agent does not sufficiently describe the real-world environment. We proposed a methodology to explicitly handle noise induced by this representation mismatch, as well as noise from low-precision oracle feedback. Our approach achieved better performance than baseline methods when predicting blind spots in the real-world environment. We additionally showed that this learned model helped to avoid costly mistakes during real-world execution, while also drastically reducing the number of oracle queries. We discussed the biases caused by different types of feedback (namely, demonstrations and corrections), and assessed the benefits of each based on domain characteristics. Finally, we included an analysis of how our model performed when optimality assumptions were relaxed and added data augmentation improvements to enable more accurate blind spot predictions when using corrections and demonstrations data. Further investigations are necessary for ideal integration of blind spot models into oracle-in-the-loop execution by trading off the cost of a mistake with the cost of querying an oracle. We also noted the possibility of moving beyond a heavy reliance upon high-quality training data via active learning approaches that can obtain more informative feedback from the oracle.

References

Abel, D., Arumugam, D., Lehnert, L., & Littman, M. (2018). State abstractions for lifelong reinforcement learning. In *International Conference on Machine Learning*, pp. 10–19.

Ammar, H. B., Eaton, E., Luna, J. M., & Ruvolo, P. (2015a). Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. In *International Joint Conference on Artificial Intelligence*, pp. 3345–3351.

Ammar, H. B., Tutunov, R., & Eaton, E. (2015b). Safe policy search for lifelong reinforcement learning with sublinear regret. In *International Conference on Machine Learning*, pp. 2361–2369.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., & Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, *57*(5), 469–483.

Athalye, A., & Sutskever, I. (2017). Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*.

Barreto, A., Munos, R., Schaul, T., & Silver, D. (2016). Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*.

Barrett, S., Taylor, M. E., & Stone, P. (2010). Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*.

Berkenkamp, F., Turchetta, M., Schoellig, A., & Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pp. 908–918.

Bıyık, E., & Sadigh, D. (2018). Batch active preference-based learning of reward functions. *arXiv preprint arXiv:1810.04303*.

Blouvshtein, L., & Cohen-Or, D. (2018). Outlier detection for robust multi-dimensional scaling. *arXiv preprint arXiv:1802.02341*.

Bodesheim, P., Freytag, A., Rodner, E., Kemmler, M., & Denzler, J. (2013). Kernel null space methods for novelty detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3374–3381.

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., et al. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4243–4250. IEEE.

Brunskill, E., & Li, L. (2014). Pac-inspired option discovery in lifelong reinforcement learning. In *International Conference on Machine Learning*, pp. 316–324.

Campbell, C., & Bennett, K. P. (2001). A linear programming approach to novelty detection. In *Advances in neural information processing systems*, pp. 395–401.

Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E., Assent, I., & Houle, M. E. (2016). On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, *30*(4), 891–927.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 15.

Chang, A., Dai, A., Funkhouser, T., Halber, M., Nießner, M., Savva, M., Song, S., Zeng, A., & Zhang, Y. (2017). Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*.

Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., & Zaremba, W. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*.

Cohn, D. A., Ghahramani, Z., & Jordan, M. I. (1996). Active learning with statistical models. *Journal of artificial intelligence research*, *4*, 129–145.

Dai, W., Chen, Y., Xue, G.-R., Yang, Q., & Yu, Y. (2009). Translated learning: Transfer learning across different feature spaces. In *Advances in neural information processing systems*, pp. 353–360.

Dawid, A. P., & Skene, A. M. (1979). Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied statistics*, 20–28.

Diuk, C., Cohen, A., & Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 240–247.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., & Koltun, V. (2017). Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*.

Frye, C., & Feige, I. (2019). Parenting: Safe reinforcement learning from human input. *arXiv preprint arXiv:1902.06766*.

Gal, Y. (2016). *Uncertainty in deep learning*. Ph.D. thesis, PhD thesis, University of Cambridge.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059.

Garcıa, J., & Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, *16*(1), 1437–1480.

Griffith, S., Subramanian, K., Scholz, J., Isbell, C., & Thomaz, A. L. (2013). Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2625–2633.

Gupta, M., Gao, J., Aggarwal, C. C., & Han, J. (2014). Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, *26*(9), 2250–2267.

He, Z., Julian, R., Heiden, E., Zhang, H., Schaal, S., Lim, J., Sukhatme, G., & Hausman, K. (2018). Zero-shot skill composition and simulation-to-real transfer by learning task representations. *arXiv preprint arXiv:1810.02422*.

Hoffmann, H. (2007). Kernel pca for novelty detection. *Pattern recognition*, *40*(3), 863–874.

Isele, D., Luna, J. M., Eaton, E., Gabriel, V., Irwin, J., Kallaher, B., & Taylor, M. E. (2016a). Lifelong learning for disturbance rejection on mobile robots. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pp. 3993–3998. IEEE.

Isele, D., Rostami, M., & Eaton, E. (2016b). Using task features for zero-shot knowledge transfer in lifelong learning.. In *IJCAI*, pp. 1620–1626.

Joshi, A. J., Porikli, F., & Papanikolopoulos, N. (2009). Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2372–2379. IEEE.

Julian, R., Heiden, E., He, Z., Zhang, H., Schaal, S., Lim, J., Sukhatme, G., & Hausman, K. (2018). Scaling simulation-to-real transfer by learning composable robot skills. *arXiv preprint arXiv:1809.10253*.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237–285.

Kahn, G., Villaflor, A., Pong, V., Abbeel, P., & Levine, S. (2017). Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*.

Kapoor, A., Grauman, K., Urtasun, R., & Darrell, T. (2007). Active learning with gaussian processes for object categorization. In *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8. IEEE.

Kendall, A., Badrinarayanan, V., & Cipolla, R. (2015). Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*.

Knox, W. B., & Stone, P. (2009). Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pp. 9–16. ACM.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., & Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 3675–3683.

Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

Lakkaraju, H., Kamar, E., Caruana, R., & Horvitz, E. (2017). Identifying unknown unknowns in the open world: Representations and policies for guided exploration.. In *AAAI*, pp. 2124–2132.

Lütjens, B., Everett, M., & How, J. P. (2018). Safe reinforcement learning with model uncertainty estimates. *arXiv preprint arXiv:1810.08700*.

Malinin, A., & Gales, M. (2018). Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pp. 7047–7058.

Markou, M., & Singh, S. (2003a). Novelty detection: a reviewpart 1: statistical approaches. *Signal processing*, *83*(12), 2481–2497.

Markou, M., & Singh, S. (2003b). Novelty detection: a reviewpart 2:: neural network based approaches. *Signal processing*, *83*(12), 2499–2521.

McAllister, R., Gal, Y., Kendall, A., Van Der Wilk, M., Shah, A., Cipolla, R., & Weller, A. V. (2017). Concrete problems for autonomous vehicle safety: advantages of bayesian deep learning.. International Joint Conferences on Artificial Intelligence, Inc.

Munos, R., Stepleton, T., Harutyunyan, A., & Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062.

Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pp. 4026–4034.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE.

Peng, X. B., Berseth, G., Yin, K., & Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, *36*(4), 41.

Prakash, B., Khatwani, M., Waytowich, N., & Mohsenin, T. (2019). Improving safety in reinforcement learning using model-based architectures and human intervention. *arXiv preprint arXiv:1903.09328*.

Radovanović, M., Nanopoulos, A., & Ivanović, M. (2015). Reverse nearest neighbors in unsupervised distance-based outlier detection. *IEEE transactions on knowledge and data engineering*, *27*(5), 1369–1382.

Ramakrishnan, R., Kamar, E., Dey, D., Shah, J., & Horvitz, E. (2018). Discovering blind spots in reinforcement learning. *Proceedings of the 17th international joint conference on Autonomous agents and multiagent systems*.

Ramakrishnan, R., Kamar, E., Nushi, B., Dey, D., Shah, J., & Horvitz, E. (2019). Overcoming blind spots in the real world: Leveraging complementary abilities for joint execution..

Ramakrishnan, R., Narasimhan, K., & Shah, J. (2016). Interpretable transfer for reinforcement learning based on object similarities. In *Proceedings of the IJCAI Interactive Machine Learning Workshop*. sn.

Ramakrishnan, R., Zhang, C., & Shah, J. (2017). Perturbation training for human-robot teams. *Journal of Artificial Intelligence Research*, *59*, 495–541.

Roy, N., & McCallum, A. (2001). Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown*, 441–448.

Ruderman, A., Everett, R., Sikder, B., Soyer, H., Uesato, J., Kumar, A., Beattie, C., & Kohli, P. (2018). Uncovering surprising behaviors in reinforcement learning via worst-case analysis..

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., & Hadsell, R. (2016). Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*.

Ruvolo, P., & Eaton, E. (2013). Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pp. 507–515.

Saunders, W., Sastry, G., Stuhlmueller, A., & Evans, O. (2017). Trial without error: Towards safe reinforcement learning via human intervention. *arXiv preprint arXiv:1707.05173*.

Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., & Platt, J. C. (2000). Support vector method for novelty detection. In *Advances in neural information processing systems*, pp. 582–588.

Shah, A., Kamath, P., Shah, J. A., & Li, S. (2018a). Bayesian inference of temporal task specifications from demonstrations. In *Advances in Neural Information Processing Systems*, pp. 3804–3813.

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2018b). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pp. 621–635. Springer.

Shaham, U., Yamada, Y., & Negahban, S. (2015). Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*.

Sheshadri, A., & Lease, M. (2013). Square: A benchmark for research on computing crowd consensus. In *First AAAI Conference on Human Computation and Crowdsourcing*.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Stadie, B. C., Abbeel, P., & Sutskever, I. (2017). Third-person imitation learning. *arXiv preprint arXiv:1703.01703*.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Taylor, M. E., Kuhlmann, G., & Stone, P. (2008). Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pp. 283–290. International Foundation for Autonomous Agents and Multiagent Systems.

Taylor, M. E., & Stone, P. (2007). Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pp. 879–886. ACM.

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(Jul), 1633–1685.

Taylor, M. E., Stone, P., & Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, *8*(Sep), 2125–2167.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., & Mannor, S. (2017). A deep hierarchical approach to lifelong learning in minecraft.. In *AAAI*, Vol. 3, p. 6.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pp. 23–30. IEEE.

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2017). Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*.

Tripathy, R. K., & Bilionis, I. (2018). Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, *375*, 565–588.

Uesato, J., Kumar, A., Szepesvari, C., Erez, T., Ruderman, A., Anderson, K., Heess, N., Kohli, P., et al. (2018). Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. *arXiv preprint arXiv:1812.01647*.

van Baar, J., Corcodel, R., Sullivan, A., Jha, D., Romeres, D., & Nikovski, D. (2018). Simulation to real transfer learning with robustified policies for robot tasks..

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., & Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*.

Wymann, B., Espié, E., Guionneau, C., Dimitrakakis, C., Coulom, R., & Sumner, A. (2000). Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, *4*, 6.

Zadrozny, B., Langford, J., & Abe, N. (2003). Cost-sensitive learning by cost-proportionate example weighting. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pp. 435–442. IEEE.

Zhang, F., Leitner, J., Ge, Z., Milford, M., & Corke, P. (2017). Adversarial discriminative sim-to-real transfer of visuo-motor policies. *arXiv preprint arXiv:1709.05746*.

Zhang, Z., Deng, J., Marchi, E., & Schuller, B. (2013). Active learning by label uncertainty for acoustic emotion recognition. In *Proceedings INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France*.

Zhu, J., Wang, H., Tsou, B. K., & Ma, M. (2009). Active learning with sampling by uncertainty and density for data annotations. *IEEE Transactions on audio, speech, and language processing*, *18*(6), 1323–1331.