BENCHMARKING CONSTRAINT INFERENCE IN INVERSE REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

When deploying Reinforcement Learning (RL) agents into a physical system, we must ensure that these agents are well aware of the underlying constraints. In many real-world problems, however, the constraints followed by expert agents (e.g., humans) are often hard to specify mathematically and unknown to the RL agents. To tackle these issues, Inverse Constrained Reinforcement Learning (ICRL) considers the formalism of Constrained Markov Decision Processes (CMDPs) and estimates constraints from expert demonstrations by learning a constraint function. As an emerging research topic, ICRL does not have common benchmarks, and previous works tested their algorithms with hand-crafted environments (e.g., grid worlds). In this paper, we construct an ICRL benchmark in the context of two major application domains: robot control and autonomous driving. For each environment, we design relevant constraints, generate the corresponding expert trajectories, and empirically justify the importance of these constraints. To recover the constraints from expert demonstrations, previous ICRL methods typically learn a deterministic constraint function, which might dismiss the true constraint during training. We tackle this issue by proposing a variational Bayesian approach to model the posterior distribution of candidate constraints. Empirical evaluation shows this method outperforms other baselines in terms of collecting rewards and satisfying constraints. The benchmark, including the instructions for reproducing ICRL algorithms, is available at *temporarily hidden due to the anonymous policy*.

1 INTRODUCTION

Constrained Reinforcement Learning (CRL) algorithms typically learn a policy under some known or predefined constraints (Liu et al., 2021).. This setting, however, is not realistic in many real-world problems since it is difficult to specify the exact constraints that an agent should follow, especially when these constraints are time-varying, context-dependent and inherent to experts' own experience, and further, such information may not be completely revealed to the agent. For example, human drivers tend to determine an implicit speed limit and a minimum gap to other cars based on the traffic conditions, rules of the road, weather and social norms. To derive a driving policy that matches human performance, an autonomous agent needs to infer these constraints from expert demonstrations.

An important approach to recovering the underlying constraints is Inverse Constrained Reinforcement Learning (ICRL) Malik et al. (2021). ICRL infers a constraint function (or sets for discrete space) to approximate constraints respected by expert demonstrations. This is often done by alternating between updating an imitating policy and a constraint function. Figure 1 summarizes the main procedure of ICRL. As an emerging research topic, ICRL does not have common datasets and benchmarks for evaluation. Existing validation methods heavily depend on the safe-Gym Ray et al. (2019) and grid-world environments. Utilizing these environments has some important drawbacks: 1) These environments are designed for control instead of constraint inference. To fill this gap, previous works often pick some environments and add hand-crafted constraints to them. Striving for simplicity, many of the selected environments have discretized state and action spaces of limited size Scobee & Sastry (2020); McPherson et al. (2021); Glazier et al.

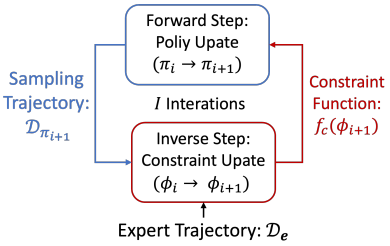

Figure 1: The flowchart of ICRL.

(2021); Papadimitriou et al. (2021); Gaurav et al. (2022). Generalizing model performance in these simple environments to practical applications will be difficult. 2) These environments do not include expert demonstrations, and thus previous works manually generate their own trajectories. Since the quality of expert trajectories can significantly influence model performance, it is difficult to compare different ICRL algorithms without a consistent expert dataset.

In this paper, we propose a benchmark for evaluating Inverse Constrained Reinforcement Learning (ICRL) algorithms. This benchmark includes both virtual environments and realistic environments. The virtual environments are based on MuJoCo Todorov et al. (2012), but we update some of these robot control tasks by modifying reward functions and including specific constraints. The realistic environments are constructed based on a highway vehicle tracking dataset Krajewski et al. (2018), so the environments can suitably reflect what happens in the realistic driving scenario. We consider constraints about car velocities and distances in these realistic environments. Each of these environments is accompanied by a demonstration dataset generated by an expert agent knowing the ground-truth constraints. To demonstrate that the added constraints are important for control, we empirically compare the performance of agents trained with and without the constraints.

For ease of comparison, our benchmark includes the state-of-the-art ICRL baselines. These baselines learn a deterministic constraint function that is most likely to differentiate expert trajectories (from $\mathcal{D}_e$ in Figure 1) from the generated ones (from $\mathcal{D}_{\pi_i}$) under some popular framework (e.g., Maximum Entropy Ziebart et al. (2008)), which, however, is often at the cost of dismissing the *true* constraint for temporally not possessing the most desirable qualities Chan & van der Schaar (2021). On the other hand, a more conceptionally-satisfying method is accounting for all possibilities of the learned constraint by modeling its posterior distribution. To extend this Bayesian approach to solve the task in our benchmark, we propose a Variational Inverse Constrained Reinforcement Learning (VICRL) algorithm that can efficiently infer constraints from the environment with a high-dimensional and continuous state space. We demonstrate the performance of VICRL by comparing its rewards and constraint violation rate with other ICRL baselines in our virtual and realistic environments. Empirical results show that the policies learned with the VICRL constraints can outperform others in terms of constraint satisfaction and reward accumulation.

2 Related Work

Inferring Constraints from Demonstrations. Previous works commonly inferred constraints to identify whether an action is allowed or a state is safe. Among these works, (Chou et al., 2018; Scobee & Sastry, 2020; McPherson et al., 2021; Park et al., 2019) are based on the *discrete* state-action space and constructed constraint sets to distinguish feasible state-action pairs from infeasible ones. Regarding *continuous* domains, the goal is to infer the boundaries between feasible and infeasible state-action pairs: (Lin et al., 2015; 2017; Armesto et al., 2017) estimated a constraint matrix from observations based on the projection of its null-space matrix. (Pérez-D'Arpino & Shah, 2017) learned geometric constraints by constructing a knowledge base from demonstration. (Menner et al., 2021) proposed to construct constraint sets that correspond to the convex hull of all observed data. (Malik et al., 2021; Gaurav et al., 2022) approximated constraints by learning neural functions from demonstrations. Some previous works (Calinon & Billard, 2008; Ye & Alterovitz, 2011; Pais et al., 2013; Li & Berenson, 2016; Mehr et al., 2016) focused on learning local trajectory-based constraints from a single trajectory. These works focus on inferring *deterministic* constraints while some recent works have learned *stochastic* constraints, for example, (Glazier et al., 2021) learned probabilistic constraints by assuming the environment constraint follows a logistic distribution. (Chou et al., 2020; Papadimitriou et al., 2021) utilized a Bayesian approach to update their belief over constraints, but these methods are restricted to discrete state spaces or toy environments like grid-world.

Testing Environments for ICRL. To the best of our knowledge, there is no common benchmark for ICRL, and thus previous works often define their own environments for evaluation, including 1) *Grid-World* is the most popular environment due to its simplicity and interpretability. Previous works (Scobee & Sastry, 2020; McPherson et al., 2021; Papadimitriou et al., 2021; Glazier et al., 2021; Gaurav et al., 2022) added some obstacles to the grid map and examined whether their algorithms can locate these obstacles by observing expert demonstrations, but it is difficult to generalize the model performance in these gird worlds to real applications with high-dimensional and continuous state spaces. 2) *Robotic Applications* have been used as the test bed for constraint inference, for example, the manipulation of robot arms (Park et al., 2019; Menner et al., 2021; Armesto et al., 2017;

Pérez-D'Arpino & Shah, 2017), quadrotors (Chou et al., 2019; 2020), and humanoid robot hands (Lin et al., 2017). However, there is no consistent type of robot for comparison, and the corresponding equipment is not commonly available. A recent work (Malik et al., 2021) used a *robotic simulator* by adding some pre-defined constraints into the simulated environments. Our virtual environments use a similar setting, but we cover more control tasks and include a detailed study of the environments and the added constraints. 3) *Safety-Gym (Ray et al., 2019)* is one of the most similar benchmarks to our work. However, Safety Gym is designed for validating the *forward* policy-updating algorithm given the constraints, whereas our benchmark is designed for the *inverse* constraint-inference problem.

3 BACKGROUND

In this section, we introduce Inverse Constrained Reinforcement Learning (ICRL) that learns constraint functions by iteratively solving both a forward Constrained Reinforcement Learning problem (CRL) and an inverse constraint inference problem (see Figure 1).

3.1 CONSTRAINED REINFORCEMENT LEARNING

Constrained Reinforcement Learning (CRL) is based on Constrained Markov Decision Processes (CMDPs) $\mathcal{M}^c$, which can be defined by a tuple $(\mathcal{S}, \mathcal{A}, p_{\mathcal{R}}, p_{\mathcal{T}}, \{(p_{\mathcal{C}_i}, \epsilon_i)\}_{\forall i}, \gamma, T)$ where: 1) $\mathcal{S}$ and $\mathcal{A}$ denote the space of states and actions. 2) $p_{\mathcal{T}}(s'|s,a)$ and $p_{\mathcal{R}}(r|s,a)$ define the transition and reward distributions. 3) $p_{\mathcal{C}_i}(c|s,a)$ denotes a stochastic constraint function with an associated bound ϵ_i, where i indicates the index of a constraint. 4) $\gamma \in [0,1)$ is the discount factor and h is the planning horizon (usually infinite). The goal of CRL is to find a (deterministic or stochastic) policy π that maximizes expected discounted rewards under a set of cumulative soft constraints:

$$\arg\max_{\pi} \mathbb{E}_{p_{\mathcal{R}}, p_{\mathcal{T}}, \pi}\left[\sum_{t=0}^{h} \gamma^t r_t\right] \text{ s.t. } \mathbb{E}_{p_{\mathcal{C}_i}, p_{\mathcal{T}}, \pi}\left[\sum_{t=0}^{h} \gamma^t c_{i,t}\right] \leq \epsilon_i \ \forall i \in [0, I] \tag{1}$$

In the above formulation, constraints consist of bounds on the expectation of cumulative constraint values. In practice, it is often more desirable to express bounds on the cumulative constraint value of a single trajectory instead of the expectation of all trajectories. Hence, we consider a more restrictive objective that requires all the sampled trajectories to satisfy the constraints:

$$\arg\max_{\pi} \mathbb{E}_{p_{\mathcal{R}}, p_{\mathcal{T}}, \pi}\left[\sum_{t=0}^{h} \gamma^t r_t\right] \text{ s.t. } \sum_{t=0}^{h} \gamma^t c_{i,t} \leq \epsilon_i \ \forall i \in [0, I] \tag{2}$$

where $c_{i,t} \sim p_{\mathcal{C}_i}(c|s_t, a_t)$ and $(s_t, a_t) \sim p_{\mathcal{T}}(s_t|s_{t-1}, a_{t-1})\pi(a_t|s_t)$. This formulation effectively describes hard constraints for each trajectory that may be difficult to satisfy in stochastic environments. Nevertheless, hard constraints are natural and often desired by practitioners. Hence, we will report the constraint violation rate (i.e., the probability with which a policy violates a constraint in a trajectory) in the experiments (see Section 4 and Section 6). Ultimately, we can think of this setup as a multi-objective problem where agents seek to maximize expected rewards while minimizing the constraint violation rate. Note that CRL problems often assume the constraint signals are directly observable from the environment, but in real-world problems, such constraint signals are not readily available, and we must infer these constraints from the environment by solving the following inverse problem.

3.2 INVERSE CONSTRAINT INFERENCE

In practice, instead of observing the constraint signals, we often have access to expert demonstrations that follow the underlying constraints. Under this setting, the agent must recover the constraint models from the dataset. This is a challenging task since there might be various equivalent combinations of reward distributions and constraints that can explain the same expert demonstrations (Ziebart et al., 2008; Malik et al., 2021). To simplify the problem, ICRL algorithms generally assume that rewards are observable and the goal is to recover only the constraints. This is the key difference with Inverse Reinforcement Learning (IRL) whose goal is to learn rewards from an unconstrained MDP. Although the goals are different, ICRL and IRL both solve the problems of learning from demonstrations and thus have many connections, for example, our baselines cover previous works (Scobee & Sastry, 2020) that utilized the Maximum Entropy IRL framework for constraint inference. Besides, IRL algorithms often assume that the expert agents are optimal or near-optimal in terms of maximizing the rewards. ICRL further assumes that the agents must satisfy underlying constraints. How to relax these assumptions will be an important direction of future work.

4 BENCHMARK

The goal of ICRL is to discover constraints from demonstration data. However, the *true* constraints in real-world datasets are often unknown, which makes it difficult to quantify the quality of *learned* constraints. To solve these issues, we develop an ICRL benchmark that incorporates external constraints to different environments. This benchmark facilitates the development of ICRL algorithms by 1) providing both virtual and realistic environments for evaluation and 2) enabling the quantification of model performance with some common RL metrics. For ease of application, we introduce some predefined constraints (see Table 1), generate the corresponding expert datasets, and empirically demonstrate their effectiveness under these environments.

4.1 VIRTUAL ENVIRONMENT

An important application of RL is robotic control, and our virtual benchmark mainly studies the robot control task with a location constraint. In practice, this type of constraint captures the locations of obstacles in the environment. For example, the agent observes that none of the expert agents visited some places in the environment. Then it is reasonable to infer that these locations must be unsafe, which can be represented by constraints. Although the real-world tasks might require more complicated constraints, our benchmark, as the first benchmark for ICRL, could serve as a stepping stone for these tasks.

Environment Settings. We implement our virtual environments by utilizing MuJoCo Todorov et al. (2012), a virtual simulator suited to robotic control tasks. To extend MuJoCo for constraint inference, we modify the MuJoCo environments by incorporating some predefined constraints into each environment and adjusting some reward terms. Table 1 summarizes the environment settings (see Appendix A.1 for more details). The virtual environments have 5 different robotic control environments simulated by MuJoCo. We add constraints on the X-coordinate of these robots: 1) For the environments where it is relatively easier for the robot to move backward rather than forward (e.g., Half-Cheetah, Ant, and Walker), our constraints bound the robot in the forward direction (with a large X-coordinate), 2) For the environments where moving forward is easier (e.g., Swimmer), the constraint bounds the robot in the backward direction (with a small X-coordinate). In these environments, the rewards are determined by the distance that a robot moves between the current and the previous time step, so the robot is likely to violate the constraints in order to maximize the magnitude of total rewards (for which we show in Section 4.4). To increase difficulty, we include an additional Biased Pendulum environment that has a larger reward on the left side. We nevertheless enforce a constraint to prevent the agent to go too far on the left side. The agent must resist the influence of high rewards and stay in safe regions.

Table 1: The virtual and realistic environments in our benchmark.

Type	Name	Dynamics	Obs. Dim.	Act. Dim.	Constraints
Virtual	Blocked Half-cheetah	Deterministic	18	6	X-Coordinate $\geq$ -3
	Blocked Ant	Deterministic	113	8	X-Coordinate $\geq$ -3
	Biased Pendulumn	Deterministic	4	1	X-Coordinate $\geq$ -0.015
	Blocked Walker	Deterministic	18	6	X-Coordinate $\geq$ -3
	Blocked Swimmer	Deterministic	10	2	X-Coordinate $\leq$ 0.5
Realistic	HighD Velocity Constraint	Stochastic	76	2	Car Velocity $\leq$ 40 m/s
	HighD Distance Constraint	Stochastic	76	2	Car Distance $\geq$ 20 m

Note that our virtual benchmark has not included some popular environments with discrete state and action spaces (e.g., Gird-World), for which we clarify from the following perspectives: 1) *Real-world Application*. One of the main goals of proposing this benchmark is to fill the gap between the simulated environment with the real-world application (see our introduction), and generalizing the performance of the discrete benchmark to real-world applications in continuous domain input is difficult. 2) *Optimization Goals.* Under the discrete environments (e.g., Grid-worlds), Bellman optimality can be reached by dynamic programming (an optimal policy ($\pi^* = \arg\max_a Q^*(s,a)$) can be uniquely found) Sutton & Barto (2018), whereas, in this work, our baselines utilizes function approximation for tackling continuous domains. These algorithms approximate the optimal policies by minimizing a sub-optimal upper bound (e.g., on the regret $\|\hat{Q} - Q^*\|_\infty$). There is no guarantee that our baseline can converge to the optimal policy satisfying the Bellman Optimality. The gap between continuous space and tractable discrete space is significant, which forms a tradition to evaluate the tabular-based algorithms with Grid-worlds Scobee & Sastry (2020) and the function approximation algorithms with continuous environments respectively Malik et al. (2021).

4.2 REALISTIC ENVIRONMENT

Our realistic environment defines a highway driving task. This HighD environment examines if the agent can drive safely the ego car to the destination by following the constraints learned from human drivers' trajectories (see Figure 2). In practice, many of these constraints are based on driving context and human experience. For example, human drivers tend to keep larger distances from trucks and drive slower on crowded roads. Adding these constraints to an auto-driving system can facilitate a more natural policy that resembles human preferences.

Figure 2: The Highway Driving (HighD) environment. The ego car is in blue, other cars are in red. The ego car can only observe the things within the region around (marked by blue). The goal is to drive the ego car to the destination (in yellow) without going off-road, colliding with other cars, or violating time limits and other constraints (e.g., velocity and distance to other vehicles).

Environment Settings. This environment is constructed by utilizing the HighD dataset Krajewski et al. (2018). Within each recording, HighD contains information about the static background (e.g., the shape and the length of highways), the vehicles, and their trajectories. We break these recordings into 3,041 scenarios so that each scenario contains less than 1,000 time steps. To create the RL environment, we randomly select a scenario and an ego car for control in this scenario. The game context, which is constructed by following the background and the trajectories of other vehicles, reflects the driving environment in real life. To further imitate what autonomous vehicles can observe on the open road, we ensure the observed features in our environment are commonly used for autonomous driving (e.g., velocity, and distances to nearby vehicles). These features reflect only partial information about the game context. To collect these features, we utilize the features collector from Commonroad RL Wang et al. (2021). In this HighD environment, we mainly study a car velocity constraint and a car distance constraint (see Table 1) to ensure the ego car can drive at a safe speed and keep a proper distance from other vehicles. Our experiment in Section 6 also studies an environment having both of these constraints.

Note that the HighD environment is stochastic since 1) Human drivers might behave differently under the same context depending on the road conditions and their driving preferences. The population of drivers induces underlying transition dynamics that are stochastic. The trajectories in the HighD dataset are essentially samples from these stochastic transition dynamics. 2) Each time an environment is reset (either the game ends or the step limit is reached), it randomly picks a scenario with a set of driving trajectories. This is equivalent to sampling from the aforementioned transition dynamics.

4.3 DEMONSTRATION DATASET

In this work, we generate the expert demonstration dataset for constraint inference since 1) the virtual environments do not include demonstrations and 2) utilizing the trajectories in the HighD dataset under the realistic environments is problematic. This is because the underlying constraints in human demonstrations are unknown, which makes it difficult to determine whether the ground-truth constraints are broken. To generate the dataset, we train a PPO-Lagrange (PPO-Lag) under the CMDP with the known constraints (Table 1 and Appendix A.1) by performing the following steps:

Training Expert Agent with PPO-Lag. We train expert agents based on ground-truth constraints in both virtual environments and realistic environments. The expert agent is trained by maximizing the following PPO-Lag objective based on the Reward Constrained Policy Optimization (RCPO) method Tessler et al. (2019):

$$\min_{\lambda} \max_{\pi} \mathbb{E}_{(s,a)\sim\pi}\left(\sum_{t=0}^{h}\gamma^t r(s_t,a_t)\right) + \mathcal{H}(\pi) - \lambda\left[\mathbb{E}_{(s,a)\sim\pi}\left(\sum_{t=0}^{h}\gamma^t c^*(s,a)\right) - \epsilon\right] \tag{3}$$

where $\mathcal{H}$ denotes the entropy term and c^* is the ground-truth constraint function from the environment. Note that PPO-Lag assumes the reward and the constraint functions are deterministic and the agent can learn to avoid most of the constrained state-action pairs with the Lagrangian penalty term. The empirical studies (Sections 4.4) show that PPO-Lag can achieve a satisfactory performance (although not optimal) given the ground-truth constraint function.

Generating a Dataset with Expert Agents. We initialize $\mathcal{D}_e = \{\emptyset\}$ and run the trained expert agents in both the virtual and the realistic environments. While running, we monitor whether the ground-truth constraints are violated until the game ends. If not, we record the corresponding trajectory: $\mathcal{D}_e = \mathcal{D}_e \cup \{\tau_e\}$, otherwise, we abandon this trajectory. We repeat this process until the demonstration dataset has enough trajectories. Note that PPO-Lag is **not** optimal in terms of maximizing the rewards or satisfying constraints. After filtering the generated trajectories with state-action pairs that violate constraints, the recorded data $\mathcal{D}_e$ must satisfy the added constraints. However, there is no guarantee that the maximum number of rewards is collected in $\mathcal{D}_e$. For more details, please check Appendix C. Our experiment (Section 6) shows ICRL algorithms can outperform PPO-Lag under some easier environments.

4.4 Empirical Study about Constraints.

In this section, we empirically demonstration the validity of the constraints defined in Table 1.

Constraints in the Virtual Environments. The thresholds of the constraints are determined experimentally to ensure that these constraints "matter" for solving the control problems. This is shown in Figure 3: 1) without knowing the constraints, a PPO agent tends to violate these constraints in order to collect more rewards within a limited number of time steps. 2) When we inform the agent of the ground-truth constraints (with the Lagrange method in Section 4.3), the PPO-Lag agent learns how to stay in the safe region, but the scale of cumulative rewards is likely to be compromised. Based on these observations, we can evaluate whether the ICRL algorithms have learned a satisfying constraint function by checking whether the corresponding RL agent can gather more rewards by performing feasible actions under the safe states (i.e., by checking *rewards* and the *constraint violation rate*).

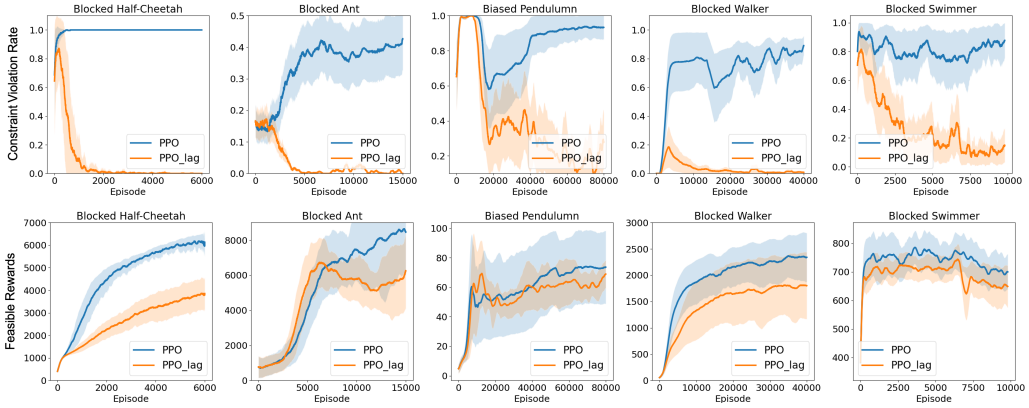

Figure 3: The constraint violation rate (top) and rewards (bottom). Environments from left to right: Blocked Half-cheetah, Blocked Ant, Biased Pendulum, Blocked Walker, and Blocked Swimmer.

Constraints in the Realistic Environments. We show the difference in performance between a PPO-Lag agent (Section 4.3) that *knows* the ground-truth constraints and a PPO agent *without knowing* the constraints. Figure 4 reports the violation rate of the speed constraint (top left) and the distance constraint (top right). The bottom graphs report the cumulative rewards in both settings. We find 1) the PPO agent tends to violate the constraints in order to get more rewards and 2) the PPO-Lag agent abandons some of these rewards in order to satisfy the constraints. Their gap demonstrates the significance of these constraints. Appendix A.5 explains why these constraints are ideal by comparing them with other candidate constraint thresholds.

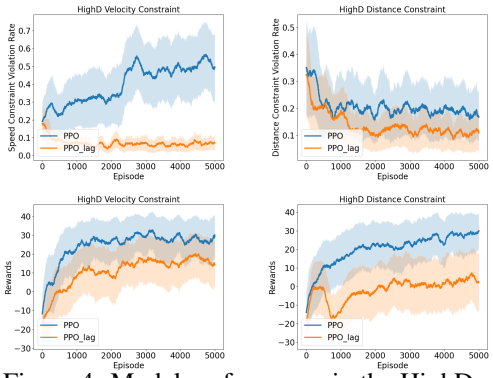

Figure 4: Model performance in the HighD environment with the velocity constraint (*left*) and the distance constraint (*right*) during *training*.

5 VARIATIONAL INVERSE CONSTRAINED REINFORCEMENT LEARNING

In this section, we introduce our VICRL for constraint inference. Unlike previous ICLR baselines that model a single constraint function, VICRL infers the distribution of candidate functions for better coverage of the true constraint. To scale Bayesian inference to high-dimensional state space, VICRL extends the variational Bayesian framework to infer constraints from the expert demonstration.

The goal of VICRL is to infer the distribution of a feasibility variable Φ so that $p(\phi|s,a)$ measures to what extent an action a should be allowed in a particular state s[1]. The instance ϕ can define a soft constraint given by: $\hat{c}_\phi(s,a) = 1-\phi$ where $\phi \sim p(\cdot|s,a)$. Given a CMDP $\mathcal{M}^{\hat{c}_\phi}$ based on the estimated constraint function $\hat{c}_\phi$, we use $p(\mathcal{D}_e|\phi)$ to define the likelihood of generating the demonstration dataset $\mathcal{D}_e$. Under this setting, the true posterior $p(\phi|\mathcal{D}_e)$ is intractable due to high-dimensional input space, so VICRL learns an approximate posterior $q(\phi|\mathcal{D}_e)$ by minimizing $\mathcal{D}_{kl}\Big[q(\phi|\mathcal{D}_e)\|p(\phi|\mathcal{D}_e)\Big]$. This is equivalent to maximizing an Evidence Lower Bound (ELBo):

$$\mathbb{E}_q\Big[\log p(\mathcal{D}_e|\phi)\Big] - \mathcal{D}_{kl}\Big[q(\phi|\mathcal{D}_e)\|p(\phi)\Big] \tag{4}$$

We introduce our method by defining the key components in this objective as follows:

The Log-Likelihood Term. We define the log-likelihood $\log\left[p(\mathcal{D}_e|\phi)\right]$ by utilizing the Maximum Entropy IRL method on a constrained MDP $\mathcal{M}^{\hat{c}_\phi}$ Malik et al. (2021); Ziebart et al. (2008):

$$\log\left[p(\mathcal{D}_e|\phi)\right] = \log\left[\frac{1}{(Z_{\mathcal{M}^{\hat{c}_\phi}})^N}\prod_{i=1}^{N}\exp\Big[r(\tau^{(i)})\Big]\,\mathbb{1}^{\mathcal{M}^{\hat{c}_\phi}}(\tau^{(i)})\right] \tag{5}$$

where 1) N denotes the number of trajectories in the demonstration dataset $\mathcal{D}_e$, 2) the normalizing term $Z_{\mathcal{M}^{\hat{c}_\phi}} = \int \exp\left[r(\tau)\right]\mathbb{1}^{\mathcal{M}^{\hat{c}_\phi}}(\tau)\mathrm{d}\tau$, and 3) the trajectory identifier $\mathbb{1}^{\mathcal{M}^{\hat{c}_\phi}}(\tau^{(i)})$ can be defined by $\phi(\tau^{(i)}) = \prod_{t=1}^{T}\phi_t$ and $\phi_t \sim p(\phi|s_t^i, a_t^i)$, which defines to what extent the trajectory $\tau^{(i)}$ is feasible. Since Φ is a continuous variable with range $[0,1]$, we parameterize $p(\phi|s,a)$ by a Beta distribution:

$$\phi(s,a) \sim p(\phi|s,a) = \text{Beta}(\alpha,\beta) \text{ where } [\alpha,\beta] = \log[1+\exp(f(s,a))] \tag{6}$$

where f is implemented by a multi-layer network with 2-dimensional outputs (for α and β).

For simplicity, we slightly overload the symbols by using $\prod_{t=1}^{T}\phi(s_t^i, a_t^i)$ to denote the trajectory identifier $\mathbb{1}^{\mathcal{M}^c}(\tau^{(i)})$ and substitute it in Equation (5). In this way, we can define:

$$\log\left[p(\mathcal{D}_e|\phi)\right] = \sum_{i=1}^{N}\Big[r(\tau^{(i)}) + \log\prod_{t=0}^{T}\phi(s_t^{(i)}, a_t^{(i)})\Big] - N\log\int\exp[r(\hat{\tau})]\prod_{t=0}^{T}\phi(\hat{s}_t,\hat{a}_t)\mathrm{d}\hat{\tau} \tag{7}$$

The gradient of this likelihood function is given by:

$$\nabla_\phi\log\left[p(\mathcal{D}_e|\phi)\right] = \sum_{i=1}^{N}\Big[\nabla_\phi\sum_{t=0}^{T}\log[\phi(s_t^{(i)}, a_t^{(i)})]\Big] - N\mathbb{E}_{\hat{\tau}\sim\pi_{\mathcal{M}^\phi}}\Big[\nabla_\phi\sum_{t=0}^{T}\log[\phi(\hat{s}_t,\hat{a}_t)]\Big] \tag{8}$$

So maximizing $\log\left[p(\mathcal{D}_e|\phi)\right]$ is equivalent to maximizing the objective:

$$\mathcal{L}(\mathcal{D}) = \sum_{i=1}^{N}\sum_{t=0}^{T}\log[\phi(s_t^{(i)}, a_t^{(i)})] - N\mathbb{E}_{\hat{\tau}\sim\pi_{\mathcal{M}^\phi}}\Big[\sum_{t=0}^{T}\log[\phi(\hat{s}_t,\hat{a}_t)]\Big] \tag{9}$$

where $\hat{\tau}$ is sampled based on executing policy $\pi_{\mathcal{M}^{\hat{\phi}}}(\hat{\tau}) = \frac{\exp[r(\tau)]\phi(\hat{\tau})}{\int\exp[r(\tau)]\phi(\tau)\mathrm{d}\tau}$. This is a maximum entropy policy that can maximize cumulative rewards subject to $\pi_{\mathcal{M}^\phi}(\tau) = 0, \forall\sum_{(s,a)\in\tau}\hat{c}_\phi(s,a) > \epsilon$ (note that $\hat{c}_\phi(s,a) = 1-\phi_t$ as defined above). We follow Tessler et al. (2019) and learn this policy with a RCPO objective. This is equivalent to substituting c^* with the estimated $\hat{c}_\phi$ in Equation (3).

The KL Divergence. Striving for the ease of computing mini-batch gradients, we approximate $\mathcal{D}_{kl}\Big[q(\phi|\mathcal{D})\|p(\phi)\Big]$ with $\sum_{(s,a)\in\mathcal{D}}\mathcal{D}_{kl}\Big[q(\phi|s,a)\|p(\phi)\Big]$. Since both the posterior and the prior are Beta distributed, we define the KL divergence by following the Dirichlet VAE Joo et al. (2020):

$$\begin{aligned}\mathcal{D}_{kl}\Big[q(\phi|s,a)\|p(\phi)\Big] = &\log\Big(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha^0+\beta^0)}\Big) + \log\Big(\frac{\Gamma(\alpha^0)\Gamma(\beta^0)}{\Gamma(\alpha)\Gamma(\beta)}\Big) \\ &+ (\alpha-\alpha^0)\Big[\psi(\alpha)-\psi(\alpha+\beta)\Big] + (\beta-\beta^0)\Big[\psi(\beta)-\psi(\alpha+\beta)\Big]\end{aligned} \tag{10}$$

[1]We use a uppercase letter and a lowercase letter to define a random variable and an instance of this variable.

where 1) $[\alpha^0, \beta^0]$ and $[\alpha, \beta]$ are parameters from the prior and the posterior functions. 2) Γ and ψ denote the gamma and the digamma functions.

6 EMPIRICAL EVALUATION

Experiment Setting. Following (Malik et al., 2021), we evaluate the quality of a recovered constraint by whether the corresponding imitation policy can maximize the cumulative rewards with a minimum violation rate for the ground-truth constraints. We repeat each experiment with different random seeds, according to which we report the mean $\pm$ standard deviation (std) results for each studied baseline and environment. For the details of model parameters and random seeds, please see Appendix A.2.

Baselines. 1) *Maximum Entropy Constraint Learning (MECL)* is based on the Maximum Entropy IRL (Ziebart et al., 2008) framework, with which Scobee & Sastry (2020) proposed an algorithm to search for constraints that most increase the likelihood of observing expert demonstrations. This algorithm focused on only discrete state spaces. A following work (Malik et al., 2021) expanded MECL to continuous states and actions, which is consistent with the setting of our benchmark. 2) *Generative Adversarial Constraint Learning (GACL)* extends MECL by following the design of Generative Adversarial Imitation Learning (GAIL) (Ho & Ermon, 2016). where $\zeta(s, a)$ assign 0s to violating state-action pairs and 1s to satisfying ones. In order to include the learned constraints into the policy update, we follow (Malik et al., 2021) and construct a new reward $r'(s, a) = r(\cdot) + \log[\zeta(\cdot)]$. This reward will punish the violating states or actions by assigning them $-\infty$ rewards, so the learned policy tends to satisfy the constraints. 3) *Binary Classifier Constraint Learning (BC2L)* shares a similar policy update method with MECL, but instead of learning a soft constraint, it utilizes a deterministic binary classifier to differentiate expert trajectories from the generated ones, which often induces a loss of identifiability.

Performance in Virtual Environments. Figure 5 and Table 2 show the training curves and the corresponding testing performance in each virtual environment. Compared to other baseline models, we find VICRL generally performs better with lower constraint violation rates and larger cumulative rewards. This is because VICRL captures the uncertainty of constraints by modeling their distributions and requiring the agent to satisfy all the sampled constraints, which facilitates a conservative imitation policy. Although MECL and GACL outperform VICRL in the Blocked Walker and the Blocked Swimmer environments, respectively, none of these algorithms can perform consistently better than the others. Figure B.2 visualizes the constraints learned by VICRL for a closer analyse.

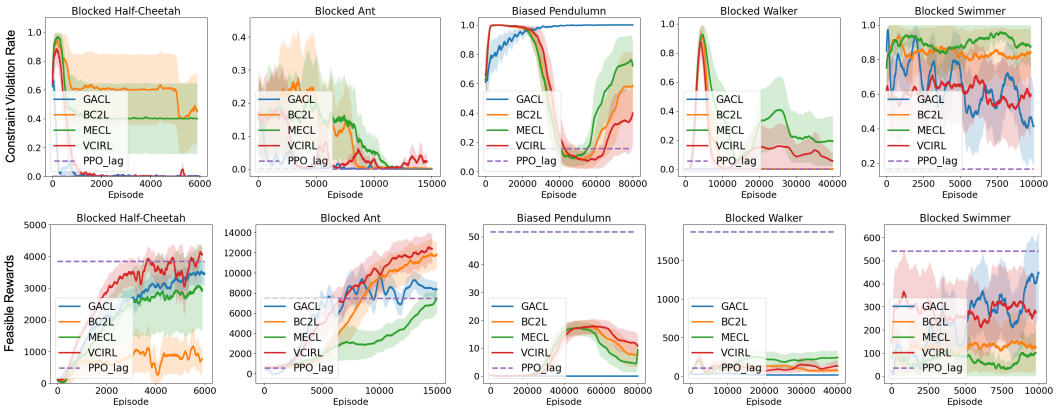

Figure 5: The constraint violation rate (top) and feasible rewards (i.e., the rewards from the trajectories without constraint violation, bottom) during training. From left to right, the environments are Blocked Half-cheetah, Blocked Ant, Bias Pendulum, Blocked Walker, and Blocked Swimmer.

Performance in Realistic Environments. Figure 7 shows the training curves and Table 2 shows the testing performance. Among the studied method, VICRL achieves a lower constraint violation rate with a satisfying number of rewards. Although GACL has the lowest violation rate, it is at the cost of significantly degrading the controlling performance, which demonstrates that directly augmenting rewards with penalties (induced by constraints) can yield a sub-optimal control policy. Appendix B.2 illustrates the causes of failures by showing the collision rate, time-out rate, and off-road rate. To illustrate how well the constraint is captured by the experimented algorithms, our plots include the

Table 2: Testing performance. We report the averaged feasible rewards and the constraint violation rate in 100 runs. * indicates the corresponding controlling performance is extremely limited, so we highlight other results. Check Appendix B.1 for the complete mean±std results. ↑ (↓) indicates that a difference is statistically greater (smaller) than the difference achieved by VICRL with p-value ≤ 0.05 according to the Wilcoxon signed-rank test. (Table 3 reports the p values.).

		Blocked Half-Cheetah	Blocked Ant	Biased Pendulum	Blocked Walker	Blocked Swimmer	HighD Velocity	HighD Distance
Feasible Rewards	GACL	3.48E+3↓	7.21E+3 ↓	8.50E-1↓	2.84E+1 ↓	**5.78E+2**↑	-1.93E+1 ↑	-1.70E+1 ↓
	BC2L	8.70E+2 ↓	1.20E+4 ↓	5.73E+0↓	4.87E+1 ↓	1.41E+2↓	-2.93E-1	3.84E+0 ↓
	MECL	3.02E+3 ↓	8.55E+3 ↓	1.02E+0↓	**1.27E+2** ↑	6.37E+1↑	**9.67E-1**	2.15E+0 ↓
	VICRL	**3.81E+3**	**1.37E+4**	**6.64E+0**	9.34E+1	1.91E+2	-8.99E-1	**4.60E+0**
Constraint Violation Rate	GACL	0%	0%	100%* ↑	0%	42% ↓	14%*	19%* ↑
	BC2L	47% ↑	0%	58%↑	0% ↓	84% ↑	33% ↑	33%
	MECL	40% ↑	0%	73% ↑	19%	88% ↑	31% ↑	41% ↑
	VICRL	0%	2%	39%	7%	59%	24%	31%

Table 3: The p Values of Wilcoxon signed-rank test across 100 runs. We repeat the test for the models trained by one random seed (we have a total of 5 random seeds) and report the averaged p values.

		Blocked Half-Cheetah	Blocked Ant	Biased Pendulum	Blocked Walker	Blocked Swimmer	HighD Velocity	HighD Distance
Feasible Rewards	GACL	1.17E-2	1.23E-06	2.46E-14	3.90E-18	3.42E-4	8.33E-5	3.28E-05
	BC2L	1.40E-15	2.98E-05	4.89E-4	2.34E-8	9.98E-10	5.75E-1	3.43E-2
	MECL	4.07E-2	3.72E-08	2.46E-14	2.94E-2	7.27E-3	3.40E-1	4.86E-2
Constraint Violation Rate	GACL	1.57E-1	6.13E-2	1.71E-18	7.68E-2	1.48E-4	2.81E-1	1.53E-2
	BC2L	7.93E-2	8.17E-1	1.84E-06	1.21E-7	8.79E-13	3.65E-2	5.54E-2
	MECL	1.52E-23	5.20E-1	1.71E-18	9.98E-2	1.55E-4	1.72E-2	7.16E-3

upper bound of rewards and the performance of the PPO-Lag agent (trained under the true constraints). It shows that there is sufficient space for future improvement under our benchmark.

To better understand the influence of environments, we explore inferring constraints from:

Stochastic Dynamics. We extend the MuJoCo environments by adding noise to the transition functions at each step such that $p(s_{t+1}|s_t, a_t) = f(s_t, a_{t)} + \eta$, where $\eta \sim \mathcal{N}(\mu, \sigma)$), Figure 6 shows the results. We find ICRL models are generally robust to additive Gaussian noises in environment dynamics until they reach a limit (e.g., $\mathcal{N}(0, 0.1)$). Another intriguing finding is that the deterministic constraint inference methods (MECL and B2CL) can benefit from a proper scale of random noise since these noisy signals facilitate a more restricted constraint function and thus a lower constraint violation rate (middle column in Figure 6).

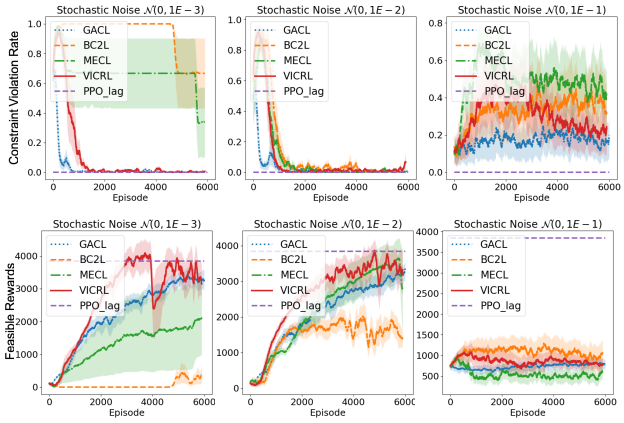

Figure 6: Model performance in the Blocked Half-Cheetah environment. From left to right, the transition function has the noises $\mathcal{N}(0, 0.001)$,$\mathcal{N}(0, 0.01)$, and $\mathcal{N}(0, 0.1)$.

Noisy Demonstration. We add noise to the expert demonstration dataset by performing *random* actions with a chance of 100%, 80%, 50%, 20% and 0% during data generation. The results in Figure 8 (left column) show the model performance, especially the rewards, is significantly influenced as we expand the scale of noise (from 0% to 100%). This is because the random actions induce sub-optimal trajectories. These trajectories are less effective in terms of reflecting the preference of expert agents, which degrades the quality of learned constraints.

Incomplete Demonstration. We randomly remove samples from the expert demonstration dataset by keeping only 30%, 50%, and 100% of the total trajectories. The results in Figure 8 (right column) show that the constraint violation rate becomes higher when we reduce the size of demonstration data.

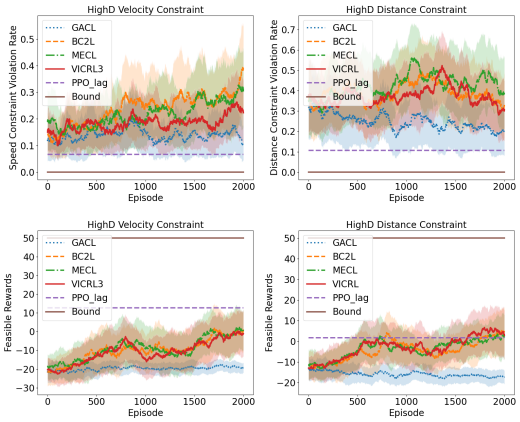

Figure 7: The constraint violation rate (top) and feasible rewards (bottom) in the HighD environments with the speed (left) and distance (right) constraints.

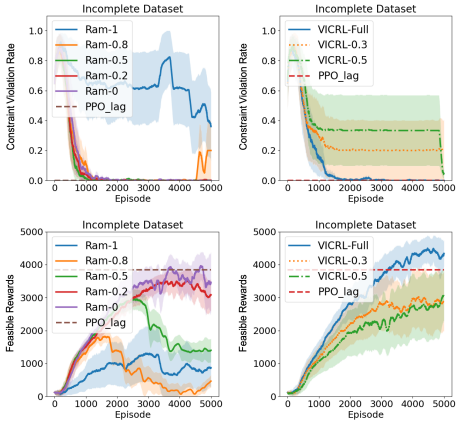

Figure 8: Performance in the Blocked HalfCheetah environment with a noisy (left column) and incomplete (right column) dataset.

This is because the expert agent commonly applies a stochastic policy, so the diversity of recorded data is influenced if we reduce the roll-out numbers (i.e., the number of trajectories).

Multi-Constraints Environment. The previously studied environments include only a true constraint. Now we study the environment that has both the speed and the distance constraints in the HighD environment. Figure 9 shows the results. Compared to the performance of its single-constraint counterparts (in Figure 7), the rewards collected by the imitation policy become significantly lower, although the constraint violation rate remains uninfluenced. Our VICRL consistently outperforms other baselines in terms of collecting rewards and respecting constraints.

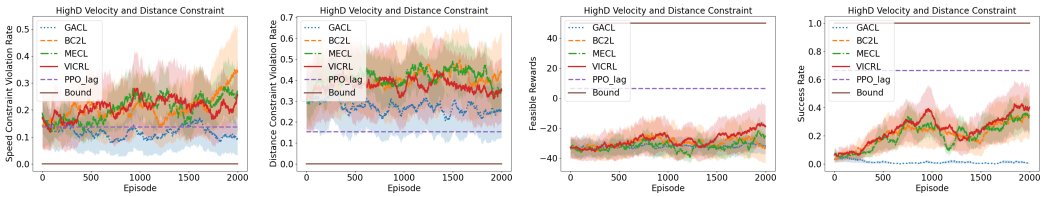

Figure 9: Model Performance in an environment with the velocity and distance constraint. From left to right, we report speed and distance constraint violation rates, feasible rewards, and success rates.

Discussion. We summarize the major challenges in our benchmark as follows:

1) *Constraint Satisfaction.* In the experiment, we observe that the learned policy may not match the expert policy even when the ground-truth constraint is recovered. This is because the Lagrange relaxation methods are sensitive to the initialization of the Lagrange multipliers and the learning rate. There is no guarantee that the policies can consistently satisfy the given constraints(See Figure 4).

2) *Noisy Demonstration Data.* ICRL algorithms typically assume that the expert demonstrations are optimal in terms of satisfying the constraints and maximizing rewards. There is no guarantee that these assumptions hold in practice since many expert agents (e.g., humans) do not always strive for optimality and constraint satisfaction. Experiment results (Figure 8) demonstrate model performance significantly degrades datasets including noisy signals.

3) *High-dimensional Features.* To learn the accurate constraint, the agent must identify exact features (e.g., velocity in x and y coordinates) among all input features (78 in total) and recover the exact constraint threshold (e.g., 40 m/s), which explains the limited model performance in the complex environment (e.g., HighD, Figure 7 and Figure 9).

7 Conclusion

In this work, we introduced a benchmark, including robot control environments and highway driving environments, for evaluating ICRL algorithms. Each environment is aligned with a demonstration dataset generated by expert agents. To extend the Bayesian approach to constraint inference, we proposed VICRL to learn a distribution of constraints. The empirical evaluation showed the performance of ICRL algorithms under our benchmark.

REFERENCES

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning (ICML)*, volume 70, pp. 22–31. PMLR, 2017.

Leopoldo Armesto, Jorren Bosga, Vladimir Ivan, and Sethu Vijayakumar. Efficient learning of constraints and generic null space policies. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1520–1526, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Daniel S. Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International Conference on Machine Learning*, volume 97, pp. 783–792, 2019a.

Daniel S. Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on Robot Learning (CoRL)*, volume 100, pp. 330–359, 2019b.

Sylvain Calinon and Aude Billard. A probabilistic programming by demonstration framework handling constraints in joint space and task space. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 367–372. IEEE, 2008.

Alex James Chan and Mihaela van der Schaar. Scalable bayesian inverse reinforcement learning. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

Letian Chen, Rohan R. Paleja, and Matthew C. Gombolay. Learning from suboptimal demonstration via self-supervised reward regression. In *Conference on Robot Learning (CoRL)*, volume 155 of *Proceedings of Machine Learning Research*, pp. 1262–1277, 2020.

Glen Chou, Dmitry Berenson, and Necmiye Ozay. Learning constraints from demonstrations. In *Workshop on the Algorithmic Foundations of Robotics, WAFR 2018*, volume 14, pp. 228–245. Springer, 2018.

Glen Chou, Necmiye Ozay, and Dmitry Berenson. Learning parametric constraints in high dimensions from demonstrations. In *Conference on Robot Learning (CoRL)*, volume 100, pp. 1211–1230, 2019.

Glen Chou, Dmitry Berenson, and Necmiye Ozay. Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations. In *Conference on Robot Learning (CoRL)*, volume 155, pp. 1612–1639, 2020.

Yinlam Chow, Ofir Nachum, Aleksandra Faust, Mohammad Ghavamzadeh, and Edgar A. Duéñez-Guzmán. Lyapunov-based safe policy optimization for continuous control. *CoRR*, abs/1901.10031, 2019.

Divyansh Garg, Shuvam Chakraborty, Chris Cundy, Jiaming Song, and Stefano Ermon. Iq-learn: Inverse soft-q learning for imitation. In *Neural Information Processing Systems (NeurIPS)*, pp. 4028–4039, 2021.

Ashish Gaurav, Kasra Rezaee, Guiliang Liu, and Pascal Poupart. Learning soft constraints from constrained expert demonstrations. *CoRR*, abs/2206.01311, 2022.

Arie Glazier, Andrea Loreggia, Nicholas Mattei, Taher Rahgooy, Francesca Rossi, and Kristen Brent Venable. Making human-like trade-offs in constrained environments by learning from demonstrations. *CoRR*, abs/2109.11018, 2021.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Neural Information Processing Systems (Neurips)*, pp. 4565–4573, 2016.

Vinamra Jain, Prashant Doshi, and Bikramjit Banerjee. Model-free IRL using maximum likelihood estimation. In *AAAI Conference on Artificial Intelligence*, pp. 3951–3958, 2019.

Weonyoung Joo, Wonsung Lee, Sungrae Park, and Il-Chul Moon. Dirichlet variational autoencoder. *Pattern Recognit.*, 107:107514, 2020.

Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. Imitation learning via off-policy distribution matching. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2020.

Robert Krajewski, Julian Bock, Laurent Kloeker, and Lutz Eckstein. The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2118–2125, 2018. doi: 10.1109/ITSC.2018.8569552.

Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. Truly batch apprenticeship learning with deep successor features. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5909–5915, 2019.

Changshuo Li and Dmitry Berenson. Learning object orientation constraints and guiding constraints for narrow passages from one demonstration. In *International Symposium on Experimental Robotics (ISER)*, volume 1, pp. 197–210, 2016.

Hsiu-Chin Lin, Matthew Howard, and Sethu Vijayakumar. Learning null space projections. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2613–2619. IEEE, 2015.

Hsiu-Chin Lin, Prabhakar Ray, and Matthew Howard. Learning task constraints in operational space formulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 309–315. IEEE, 2017.

Yongshuai Liu, Jiaxin Ding, and Xin Liu. IPO: interior-point policy optimization under constraints. In *AAAI Conference on Artificial Intelligence*, pp. 4940–4947, 2020.

Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pp. 4508–4515. ijcai.org, 2021.

Shehryar Malik, Usman Anwar, Alireza Aghasi, and Ali Ahmed. Inverse constrained reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139, pp. 7390–7399, 2021.

David Livingston McPherson, Kaylene C. Stocking, and S. Shankar Sastry. Maximum likelihood constraint inference from stochastic demonstrations. In *IEEE Conference on Control Technology and Applications, (CCTA)*, pp. 1208–1213, 2021.

Negar Mehr, Roberto Horowitz, and Anca D. Dragan. Inferring and assisting with constraints in shared autonomy. In *IEEE Conference on Decision and Control (CDC)*, pp. 6689–6696. IEEE, 2016.

Marcel Menner, Peter Worsnop, and Melanie N. Zeilinger. Constrained inverse optimal control with application to a human manipulation task. *IEEE Trans. Control. Syst. Technol.*, 29(2):826–834, 2021.

Lucia Pais, Keisuke Umezawa, Yoshihiko Nakamura, and Aude Billard. Learning robot skills through motion segmentation and constraints extraction. In *HRI Workshop on Collaborative Manipulation*, pp. 5. Citeseer, 2013.

Dimitris Papadimitriou, Usman Anwar, and Daniel S Brown. Bayesian inverse constrained reinforcement learning. In *Workshop on Safe and Robust Control of Uncertain Systems (NeurIPS)*, 2021.

Daehyung Park, Michael Noseworthy, Rohan Paul, Subhro Roy, and Nicholas Roy. Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning. In *Conference on Robot Learning (CoRL)*, volume 100, pp. 1005–1014, 2019.

Claudia Pérez-D'Arpino and Julie A. Shah. C-LEARN: learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy. In *2017 IEEE International Conference on Robotics and Automation, ICRA*, pp. 4058–4065. IEEE, 2017.

Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7:1, 2019.

Dexter R. R. Scobee and S. Shankar Sastry. Maximum likelihood constraint inference for inverse reinforcement learning. In *8th International Conference on Learning Representations, (ICLR)*. OpenReview.net, 2020.

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

Voot Tangkaratt, Bo Han, Mohammad Emtiyaz Khan, and Masashi Sugiyama. Variational imitation learning with diverse-quality demonstrations. In *International Conference on Machine Learning (ICML)*, volume 119, pp. 9407–9417, 2020.

Voot Tangkaratt, Nontawat Charoenphakdee, and Masashi Sugiyama. Robust imitation learning from noisy demonstrations. In *Artificial Intelligence and Statistics (AISTATS)*, volume 130, pp. 298–306, 2021.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations ICLR*. OpenReview.net, 2019.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pp. 5026–5033. IEEE, 2012.

Xiao Wang, Hanna Krasowski, and Matthias Althoff. Commonroad-rl: A configurable reinforcement learning environment for motion planning of autonomous vehicles. In *24th IEEE International Intelligent Transportation Systems Conference, ITSC 2021, Indianapolis, IN, USA, September 19-22, 2021*, pp. 466–472. IEEE, 2021.

Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning (ICML)*, volume 97, pp. 6818–6827, 2019.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J. Ramadge. Projection-based constrained policy optimization. In *International Conference on Learning Representations (ICLR)*, 2020.

Gu Ye and Ron Alterovitz. Demonstration-guided motion planning. In *Robotics Research - The 15th International Symposium ISRR*, volume 100, pp. 291–307, 2011.

Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, pp. 1433–1438. AAAI Press, 2008.

A More Implementation and Environment Details

A.1 More Information about the Virtual Environments

Our virtual environments are based on Mujoco (see Figure A.1). We provide more details about the virtual environments as follows:

- *Blocked Half-Cheetah.* The agent controls a robot with two legs. The reward is determined by the distance it walks between the current and the previous time step and a penalty over the magnitude of the input action. The game ends when a maximum time step (1000) is reached. We define a constraint that blocks the region with X-coordinate ≤ -3, so the robot is only allowed to move in the region with X-coordinate between -3 and ∞.

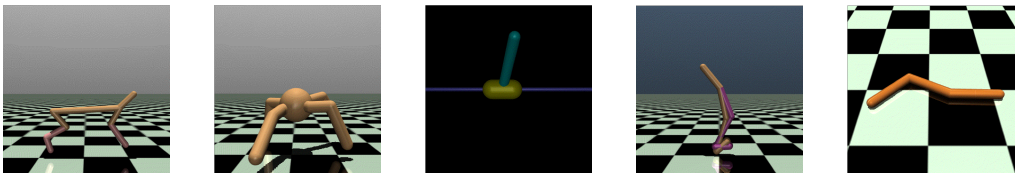

Figure A.1: Mujoco environments. From left to right, the environments are Half-cheetah, Ant, Inverted Pendulum, Walker and Swimmer.

- *Blocked Ant.* The agent controls a robot with four legs. The rewards are determined by the distance to the origin and a healthy bonus that encourages the robot to stay balanced. The game ends when a maximum time step (500) is reached. Similar to the Blocked Half-Cheetah environment, we define a constraint that blocks the region with X-coordinate ≤ -3, so the robot is only allowed to move in the region with X-coordinate between -3 and ∞.
- *Biased Pendulum.* Similar to the Gym CartPole (Brockman et al., 2016), the agent's goal is to balance a pole on a cart. The game ends when the pole falls or a maximum time step (100) is reached. At each step, the environment provides a reward of 0.1 if the X-coordinate ≥ 0 and a reward of 1 if the X-coordinate ≤ -0.01. The reward monotonically increases from 0.1 to 1 when $-0.01 <$ X-coordinate < 0. We define a constraint that blocks the region with X-coordinate ≤ -0.015, so the reward incentivizes the cart to move left, but the constraint prevents it from moving too far. If the agent can detect the ground-truth constraint threshold, it will drive the cart to move into the region with X-coordinate between -0.015 and -0.01 and stay balanced there.
- *Blocked Walker.* The agent controls a robot with two legs and learns how to make the robot walk. The reward is determined by the distance it walks between the current and the previous time step and a penalty over the magnitude of the input action (this is following the original Walker2d environment). The game ends when the robot loses its balance or reaches a maximum time step (500). Similar to the Blocked Half-Cheetah and Blocked Ant environment, we constrain the region with X-coordinate ≤ -3, so the robot is only allowed to move in the region with X-coordinate between -3 and ∞.
- *Blocked Swimmer.* The agent controls a robot with two rotors (connecting three segments) and learns how to move. The reward is determined by the distance it walks between the current and the previous time step and a penalty over the magnitude of the input action. The game ends when the robot reaches a maximum time step (500). Unlike the Blocked Half-Cheetah and Blocked Ant environment, it is easier for the Swimmer robot to move ahead than move back, and thus we constrain the region with X-coordinate ≥ 0.5, so the robot is only allowed to move in the region with X-coordinate between $-\infty$ and 0.5.

A.2 Hyper-Parameters

We published our benchmarks, including the configurations of the environments and the models, in *temporarily hidden due to the anonymous policy*. Please see the README.MD file for more details. We provide a brief summary of the hyper-parameters.

In order to develop a fair comparison among ICRL algorithms, we use the same setting for all algorithms.

In the **virtual environments**, we set 1) the batch size of PPO-Lag to 64, 2) the size of the hidden layer to 64, and 3) the number of hidden layers for the policy function, the value function, and the cost function to 3. We decide the other parameters, including the learning rate of both PPO-Lag and constraint model, by following some previous work (Malik et al., 2021) and their implementation *temporarily hidden due to the anonymous policy*. The random seeds of virtual environments are 123, 321, 456, 654, and 666.

In the **realistic environments**, we set 1) the batch size of the constraint model to 1000, 2) the size of the hidden layer to 64 and 3) the number of hidden layers for the policy function, the value function and the cost function to 3. We decide the other parameters, including the learning rate of

both PPO-Lag and constraint model, by following CommonRoad RL (Wang et al., 2021) and their implementation [2]. During our experiment, we received plenty of help from their forum [3]. We will acknowledge their help in the formal version of this paper. The random seeds of realistic environments are 123, 321, and 666.

A.3 Experimental Equipment and Infrastructures

We run the experiment on a cluster operated by the Slurm workload manager. The cluster has multiple kinds of GPUs, including Tesla T4 with 16 GB memory, Tesla P100 with 12 GB memory, and RTX 6000 with 24 GB memory. We used machines with 12 GB of memory for training the ICRL models. The number of running nodes is 1, and the number of CPUs requested per task is 16. Given the aforementioned resources, running one seed in the virtual environments and the realistic environments takes 2-4 hours and 10-12 hours respectively.

A.4 Computational Complexity

We provide a brief analysis of the computational complexity. The ICRL algorithms, including GACL, MECL, BC2L, and VICRL, use an iterative updating paradigm and thus their computational complexities are similar. Let K denote the number of iterations. Within each iteration, the algorithms update both the imitation policy and the constraint model. Let M denote the number of episodes that the PPO-Lag algorithm runs in the environments. Let N denote the number of sampling and expert trajectories. Let L denote the maximum length of each trajectory. During training, we use mini-batch gradient descent. Let B denote the batch size, and then the computational complexity is $O(KL(M+N)/B)$.

A.5 Exploring Other Constraints in the Realistic Environments

The constraint thresholds in our environments are determined empirically according to the performance (constraint violation rate and rewards) of the PPO agent and the PPO-Lag agent. To support this claim, we show the performance of other thresholds and analyze why they are sub-optimal in terms of validating ICRL algorithms.

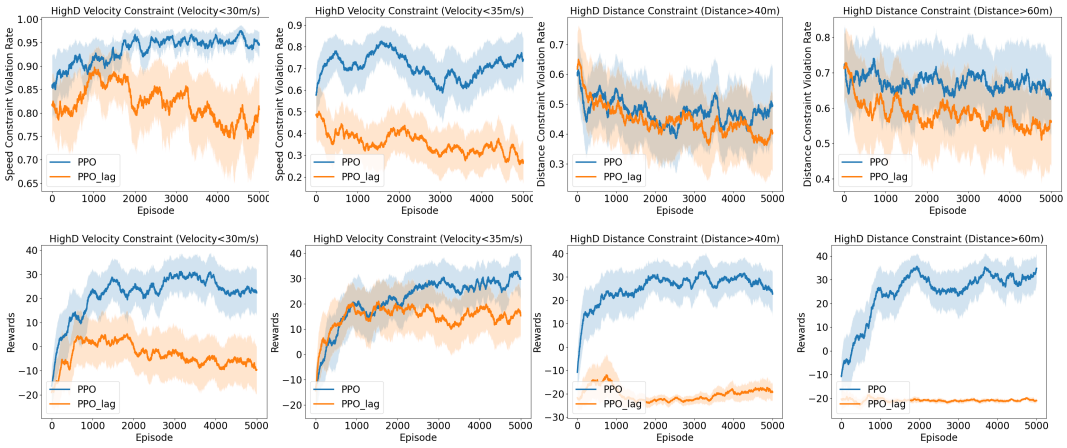

Figure A.2: From left to right, the constraint violation rate (top) and rewards (bottom) of the PPO and PPO-Lag agents in the HighD environments with constraints 1) Ego Car Velocity $<$ 30 m/s, 2) Ego Car Velocity $<$ 35 m/s, 3) Car Distance $>$ 40 m, and 4) Car Distance $>$ 60 m.

We have explored the option of using a 30m/s velocity constraint (The first column on the left in Figure A.2) and 35m/s velocity constraint (The second column on the left in Figure A.2). Ideally, these constraints should be closer to the realistic speed limit in most countries. However, the HighD

[2] *temporarily hidden due to the anonymous policy*

[3] *temporarily hidden due to the anonymous policy*

dataset comes from German highways where there is no speed limit. Moreover, when building the environment, the ego car is accompanied by an initial speed calculated from the dataset. We observed that the initial speed is already higher than the speed limit (e.g., 35m/s) in many scenarios, and thus the violation rate will always be 1 in these scenarios, leaving no opportunity for improving the policy. This explains why the corresponding violation rates are high for the PPO and the PPO-Lag agents.

We also explored the option of using a 40m distance constraint (third column in Figure A.2) and a 60m distance constraint (fourth column in Figure A.2). Ideally, these constraints should be more consistent with the 2-second gap recommendation (the average speed is around 30m/s in HighD, so the recommended gap is 2*30m/s=60m), but we find the controlling performance of the PPO-Lag agents are very limited, which shows the agent cannot even develop a satisfying control policy when knowing the ground-truth constraints. This is because the ego car learns to frequently go off-road in order to maintain the large gap.

B More Experimental Results

B.1 The Complete Results for Testing Performance

Table B.1: Testing performance in the virtual environments. We report the feasible rewards (i.e., the rewards from the trajectories without constraint violation) computed with 50 runs.

	Half-cheetah	Blocked Ant	Biased Pendulum	Blocked Walker	Blocked Swimmer
GACL	3477.53 $\pm$ 416.54	7213.62 $\pm$ 993.12	0.85 $\pm$ 0.02	28.35 $\pm$ 0.77	578.27 $\pm$ 148.16
BC2L	870.09 $\pm$ 499.03	11956.26 $\pm$ 1980.88	5.73 $\pm$ 5.60	48.73 $\pm$ 4.18	141.82 $\pm$ 152.14
MECL	3024.88 $\pm$ 1364.59	8546.19 $\pm$ 1262.03	1.02 $\pm$ 1.63	126.76 $\pm$ 52.21	63.66 $\pm$ 107.95
VICRL	3805.72 $\pm$ 511.66	13670.32 $\pm$ 2511.89	6.64 $\pm$ 4.45	93.40 $\pm$ 93.97	191.11 $\pm$ 154.57

Table B.2: Testing performance in the virtual environments. We report the constraint violation rate computed with 50 runs.

	Half-cheetah	Blocked Ant	Biased Pendulum	Blocked Walker	Blocked Swimmer
GACL	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	1.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.42 $\pm$ 0.23
BC2L	0.47 $\pm$ 0.24	0.0 $\pm$ 0.0	0.58 $\pm$ 0.23	0.0 $\pm$ 0.0	0.84 $\pm$ 0.14
MECL	0.40 $\pm$ 0.24	0.0 $\pm$ 0.0	0.73 $\pm$ 0.17	0.19 $\pm$ 0.17	0.88 $\pm$ 0.12
VICRL	0.0 $\pm$ 0.0	0.02 $\pm$ 0.02	0.39 $\pm$ 0.22	0.07 $\pm$ 0.07	0.59 $\pm$ 0.23

Table B.3: Testing performance in the realistic environments. We report the feasible rewards (i.e., the rewards from the trajectories without constraint violation) computed with 50 runs.

	HighD Velocity Constraint	HighD Distance Constraint
GACL	-19.13 $\pm$ 2.99	-17.02 $\pm$ 3.31
BC2L	-0.29 $\pm$ 11.18	3.84 $\pm$ 11.28
MECL	0.97 $\pm$ 11.48	2.15 $\pm$ 10.45
VICRL	-0.90 $\pm$ 11.80	4.60 $\pm$ 11.71

B.2 Complementary Results in the Realistic Environment

Figure B.1 reports the average velocity, collision rate, off-road rate, time-out rate and goal-reaching rate during training. We find the off-road rate of GACL is significantly higher than other methods. It explains why GACL cannot achieve a satisfying performance. Another main limitation of current baselines is their incapability of preventing the collision events, especially under the car distance constraints.

Table B.4: Testing performance in the realistic environments. We report the constraint violation rate computed with 50 runs.

	HighD Velocity Constraint	HighD Distance Constraint
GACL	0.14 ± 0.09	0.19 ± 0.11
BC2L	0.33 ± 0.15	0.33 ± 0.15
MECL	0.31 ± 0.15	0.41 ± 0.17
VICRL	0.24 ± 0.12	0.31 ± 0.15

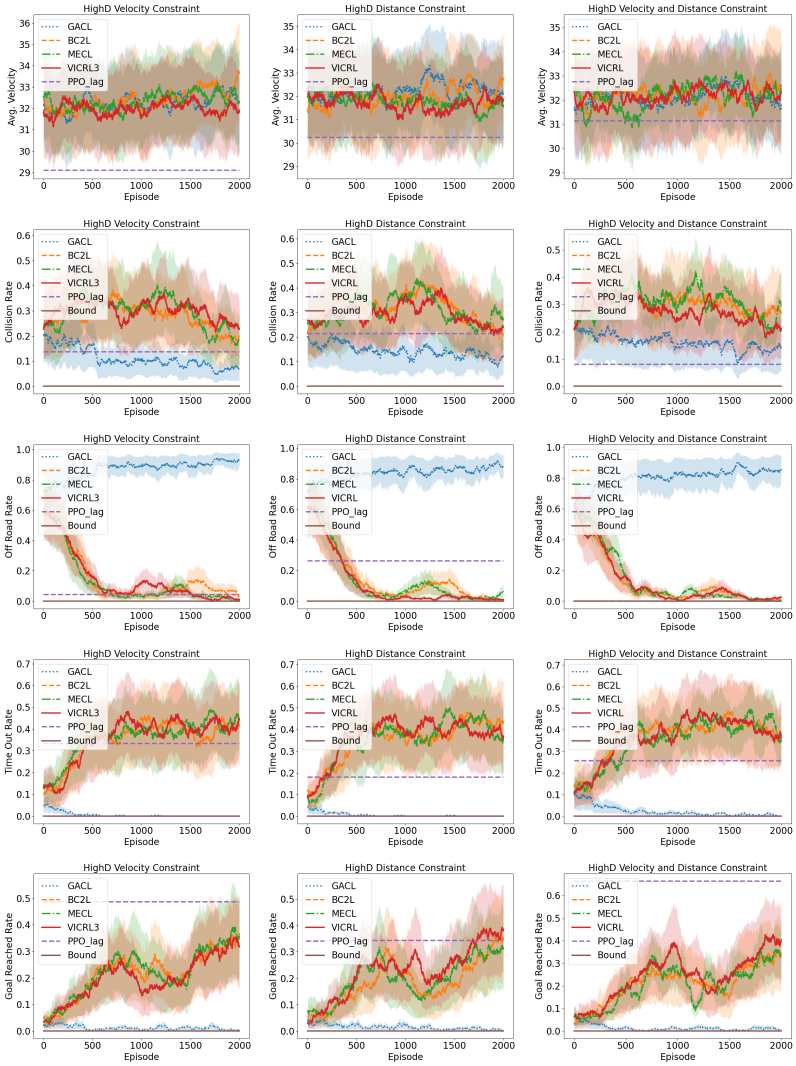

Figure B.1: The average velocity (first column), collision rate (second column), off road rate (third column), time out rate (fourth column) and goal reaching rate (last column) during training. From left to right, the environments are HighD with the velocity constraints, the distance constraints and both of these constraints.

B.3 CONSTRAINT VISUALIZATION

Figure B.2 visualizes the learned constraints with 1) partial dependency plots (red curve, on the left) accompanied by the samples from the constraint distribution (blue points, i.e., $P_{\mathcal{C}}(c|s,a)$, marked by cost) and 2) histograms (blue, on the right) showing the number of states with a specific feature value (e.g., x position) during testing. The x-axis of these plots show the features where the ground-truth constraints are defined on (this message is hidden from the agents during training). In order to understand how well the constraints are captured, we can compare these plots with the definition of ground truth constraints in Table 1.

C LIMITATIONS, CHALLENGES AND OPEN QUESTIONS

We introduce limitations and challenges in ICRL, as well as open questions for future work.

Constraint Violation. The imitation policies of ICRL agents are updated with RCPO (Tessler et al., 2019), but Lagrange relaxation methods are sensitive to the initialization of the Lagrange multipliers and the learning rate. There is no guarantee that the imitation policies can consistently satisfy the given constraints (Liu et al., 2021). As a result, even when a learned constraint function matches the ground-truth constraint, the learned policy may not match the expert policy, causing significant variation in training and sub-optimal model convergence. If we replace the Lagrange relaxation with Constrained Policy Optimization (CPO) (Achiam et al., 2017; Chow et al., 2019; Yang et al., 2020; Liu et al., 2020), ICRL may not finish training within a reasonable amount of time since CPO is computationally more expensive. How to design an efficient policy learning method that matches ICRL's iterative updating paradigm will be an important future direction.

Unrealistic Assumptions about Expert Demonstrations. ICRL algorithms typically assume that the expert demonstrations are optimal in terms of satisfying the constraints and maximizing rewards. There is no guarantee that these assumptions hold in practice since many expert agents (e.g., humans) do not always strive for optimality and constraint satisfaction. Previous works (Brown et al., 2019a;b; Wu et al., 2019; Chen et al., 2020; Tangkaratt et al., 2020; 2021), introduced IRL approaches to learn rewards from sub-optimal demonstrations, but how to extend these methods to constraint inference is unclear. A promising direction is to model *stochastic* constraints that assume that expert agents only follow the constraints with a certain probability.

Insufficient Constraint Diversity. ICRL can potentially recover complex constraints, but our benchmark mainly considers linear constraints as the ground-truth constraints (although this information is hidden from the agent). Despite this simplification, our benchmark is still very challenging: a ICRL agent must identify relevant features (e.g., velocity in x and y coordinates) among all input features (78 in total) and recover the exact constraint threshold (e.g., 40 m/s). For future work, we will explore nonlinear constraints and constraints on high-dimensional input spaces (e.g., pixels).

Online versus Offline ICRL. ICRL algorithms commonly learn an imitation policy by interacting with the environment. The online training nevertheless contradicts with the setting of many realistic applications where only the demonstration data instead of the environment is available. Given the recent progress in offline IRL (Jain et al., 2019; Lee et al., 2019; Kostrikov et al., 2020; Garg et al., 2021), extending ICRL to the offline training setting will be an important future direction.

D SOCIETAL IMPACT

Positive Societal Impacts The ability to discover what can be done and what cannot be done is an important function of modern AI systems, especially for systems that have frequent interactions with humans (e.g., house keeping robots and smart home systems). As an important stepping stone towards the design of effective systems, constraint models can help develop human-friendly AI systems and facilitate their deployments in real applications.

Negative Societal Impacts Possible real-world applications of constraint models include autonomous driving systems. Since constraint models are often represented by black-box deep models, there is no guarantee that the models are trustworthy and interpretable. When an autonomous vehicle

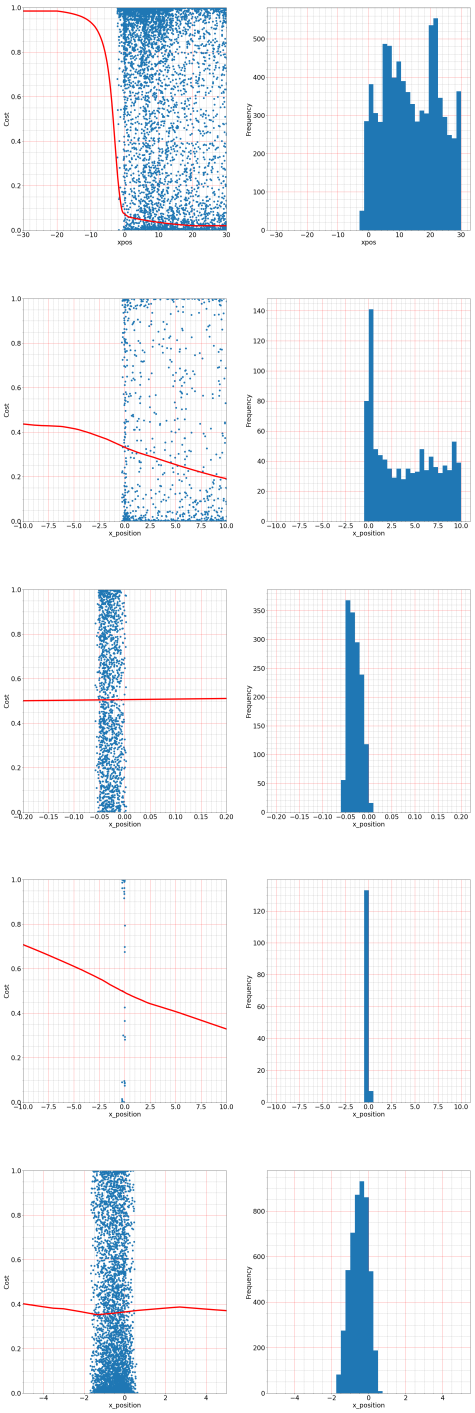

Figure B.2: Visualization the learned constraints by a pair of plots for VICRL-RS (left column) and (right column) VICRL-VaR. Each pair includes 1) partial dependency plots (red curve, on the left) accompanied by the samples from the constraint distribution (blue points, i.e., $P_{\mathcal{C}}(c|s, a)$, marked by cost) and 2) histograms (blue, on the right) showing the number of states with a specific feature value (e.g., x position) during testing. From top to bottom, the testing environments are Blocked Half-Cheetah, Blocked Ant, Biased Pendulum, Blocked Walker and Blocked Swimmer.

is involved into an accident, it is difficult to identify the cause of this accident, which might cause a loss of confidence in autonomous systems while negatively impacting society.