

# Learning Neural Control Barrier Functions from Offline Data with Conservatism

Ihab Tabbara

Department of Computer Science  
Washington University in St. Louis  
i.k.tabbara@wustl.edu

Hussein Sibai

Department of Computer Science  
Washington University in St. Louis  
sibai@wustl.edu

**Abstract:** Safety filters, particularly those based on control barrier functions, have gained increased interest as effective tools for safe control of dynamical systems. Existing correct-by-construction synthesis algorithms, however, suffer from the curse of dimensionality. Deep learning approaches have been proposed in recent years to address this challenge. In this paper, we contribute to this line of work by proposing an algorithm for training control barrier functions from offline datasets. Our algorithm trains the filter to not only prevent the system from reaching unsafe states but also out-of-distribution ones, at which the filter would be unreliable. It is inspired by Conservative Q-learning, an offline reinforcement learning algorithm. We call its outputs Conservative Control Barrier Functions (CCBFs). Our empirical results demonstrate that CCBFs outperform existing methods in maintaining safety and out-of-distribution avoidance while minimally affecting task performance.

**Keywords:** Safety filters, Control barrier functions, Offline reinforcement learning

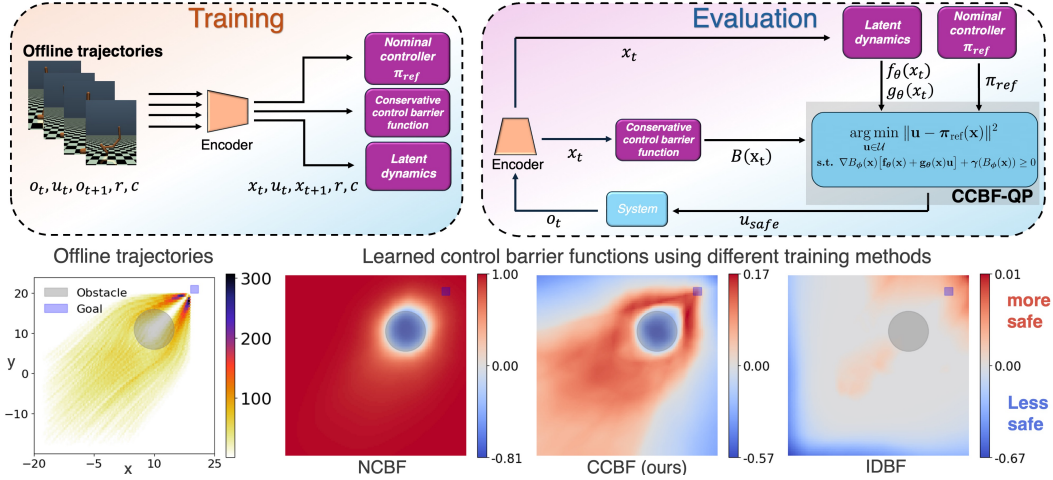


Figure 1: We propose a simple modification of the training process of neural control barrier functions when used for learning from offline data. Our learned functions can be used to define constraints in quadratic programs that can be used as safety filters that correct unsafe and out-of-distribution nominal controls. We evaluate our approach in four environments and demonstrate its improved performance over three baseline methods.

## 1 Introduction

Ensuring that autonomous systems satisfy safety specifications is critical to their deployment in various settings, such as autonomous driving [1] and robotic surgery [2]. Safety filters have emerged as promising tools for enforcing safety constraints [3]. Control barrier functions (CBF) allow the

design of efficient safety filters in the form of quadratic programs (QP) that preserve the invariance of safe sets while minimally altering nominal control [4, 3, 5]. However, synthesizing CBFs using correct-by-construction techniques, such as sum-of-squares programming [6, 7], do not scale beyond few dimensions.

To address this challenge, researchers have proposed data-driven methods for learning safety filters [8, 9, 10], which have been applied in settings with uncertain dynamics [11, 12] and high-dimensional observations, such as images [10, 13, 14, 15, 16, 17, 18] and point clouds [19, 20]. However, these filters lack the theoretical guarantees that their formally verified or synthesized counterparts have. Moreover, when deployed, by design, they will alter the nominal controls resulting in a shift between the distribution of trajectories used for their training and the ones visited during deployment. It is often assumed that the filter is trained iteratively until convergence: in each iteration, it is used to generate new trajectories whose states get labeled as safe or unsafe, and then the filter gets retrained accordingly [21]. After convergence, the distribution of states used to train the filter would match the distribution of states it will encounter during deployment. Consequently, one can use classic generalization bounds to obtain probabilistic guarantees on the correctness of the learned filter. However, in various safety-critical settings, it can be challenging and expensive to collect new trajectories, especially since they may be unsafe when generated using the filter being trained. This highlights the need for learning safety filters from offline data, while addressing or mitigating the resulting distribution shift between training and deployment.

Offline reinforcement learning (RL) has been studied extensively in the past few years [22]. Several algorithms have been proposed to learn optimal policies from offline datasets of trajectories in that literature [23, 24, 25, 26, 27, 28]. The goal is to obtain policies that outperform the ones used to generate the data or those which can be obtained using behavior cloning (BC), while accounting for distribution shift. In this paper, we aim to train safety filters using offline datasets of safe and unsafe trajectories that alter unsafe nominal controls to in-distribution (InD) safe ones without being overly restrictive. For that purpose, we introduce Conservative Control Barrier Functions (CCBFs), a method for training neural CBFs that is inspired by conservative Q-learning (CQL) [23], a prominent offline RL algorithm. The core insight is that the learned CBF should assign higher safety scores to InD states than to out-of-distribution (OOD) states, even if both are safe. This objective is similar to the CQL objective, which penalizes the overestimation of Q-values for OOD actions. By analogously discouraging high safety scores for OOD states, safety filters using CCBFs keep the system in the region of the state space in which they can provide confident estimates of safety.

We evaluate our approach against three baseline methods in four environments with unknown dynamics and with varying state dimensions. Our results show that CCBFs outperform the baselines in achieving safety without sacrificing task performance while being easy to train and not requiring significant tuning of the hyperparameters.

## 2 Related Work

**Learning safety filters from expert demonstrations** Several algorithms have been proposed recently to train neural CBFs from offline datasets [29, 30, 31, 12, 32, 8, 33, 34, 35, 36]. They differ in their objectives, their dataset generation process, their assumptions, and their guarantees, if any. Castaneda et al. [37] and Kang et al. [38] proposed learning safety filters to prevent control systems from reaching OOD states. Castaneda et al. [37] proposed an algorithm for training neural CBFs, termed in-distribution barrier functions (iDBFs), using an offline dataset of safe trajectories to prevent the control system from reaching OOD states. To train an iDBF, the algorithm first trains a BC policy using the same dataset. From each state in the training dataset, the algorithm samples random actions. If the BC policy assigns a sampled action a probability density below a user-defined threshold  $p$ , the next state reached by that action is labeled OOD and appended to the dataset. It finally uses an existing approach for training neural CBFs (e.g., [11]) to train the iDBF using the augmented dataset, treating the InD states as safe and the OOD states as unsafe. Kang et al. [38] proposed Lyapunov Density Models, combining control Lyapunov functions [39] and density models [40], which are also

learned from data to keep a control system from reaching OOD states. More recently, Yu et al. [41] proposed an algorithm for learning neural CBFs from an offline dataset consisting of a mix of labeled and unlabeled trajectories by leveraging cost-sensitive classification [42] to assign safety labels to the unlabeled demonstrations.

Several other works proposed algorithms for synthesizing CBFs from expert safe demonstrations providing formal correctness guarantees through Lipschitz continuity and coverage-based proofs [8, 43, 12]. These works assume that the dataset is sufficiently large and spread to cover the region of the state space that the system will operate in during deployment, and thus no generalization capabilities are needed from the learned filter and OOD states are not reachable.

In this paper, we aim to train neural CBFs from offline datasets of both safe and unsafe demonstrations that can be used as safety filters for control systems to avoid both unsafe and OOD states while minimally affecting task progress.

**Safe offline RL** Safe offline RL is a branch of offline RL that aims to train policies that satisfy safety constraints while maximizing reward. It tackles a similar problem to ours: given a set of trajectories of a Constrained Markov Decision Process (CMDP), design an optimal and safe policy that avoids distribution shift [44, 45]. However, it considers soft safety constraints which require the expected cost of the trajectories to remain below a user-defined threshold, in contrast with the hard constraints of avoiding unsafe states considered in our case. Several approaches incorporated safety filters into the safe offline RL pipeline [46, 47, 48]. A close work to ours is the recently proposed method FISOR [46], which similarly employs a hard state-wise constraint formulation for safe offline RL. It first adapts Hamilton Jacobi reachability to learn a value function that defines the backward reachable set of a known constraint set from the offline dataset. It uses Implicit Q-learning [24] to prevent over-optimistic safety value estimation that can result from querying OOD actions in offline RL settings. It then trains a weighted BC policy that maximizes rewards within the feasible region in the state space, i.e., the complement of the backward reachable set. We instead take a different approach and assume that the states in the dataset are labeled as safe or unsafe, train a neural CBF with an additional loss term inspired from CQL [23] to avoid over-optimistic CBF values for OOD states, and test its performance as a safety filter for various nominal controllers that are learned separately from the same dataset using various offline RL algorithms.

### 3 Preliminaries

#### 3.1 Control barrier functions

We assume that the dynamics of the systems we consider in this paper are control-affine.

**Definition 1** (Control-affine systems). *A nonlinear control-affine system can be modeled using an ordinary differential equation (ODE) of the form:*

$$\dot{x} = f(x) + g(x)u, \quad (1)$$

where  $x(t) \in \mathcal{X} \subseteq \mathbb{R}^n$  represents the state, and  $u(t) \in \mathcal{U} \subseteq \mathbb{R}^m$  denotes the control input at time  $t$ . The functions  $f : \mathcal{X} \rightarrow \mathbb{R}^n$  and  $g : \mathcal{X} \rightarrow \mathbb{R}^{n \times m}$  are assumed to be locally Lipschitz continuous.

**Definition 2** (Control barrier functions [4]). *A function  $B : \mathcal{X} \rightarrow \mathbb{R}$  is a control barrier function for system (1) if it is continuously differentiable and satisfies:*

$$\exists u \in \mathcal{U} \text{ such that } \dot{B}(x, u) + \gamma(B(x)) \geq 0, \quad \forall x \in \mathcal{D} \subseteq \mathcal{X}, \quad (2)$$

where the super-level set  $B_{\geq 0} := \{x \mid B(x) \geq 0\}$  satisfies  $B_{\geq 0} \subseteq \mathcal{D}$ , and  $\gamma : \mathbb{R} \rightarrow \mathbb{R}$  is a locally Lipschitz extended class- $\mathcal{K}_\infty$  function, meaning it is strictly increasing and satisfies  $\gamma(0) = 0$ .

A CBF  $B$  for system (1) specifies the controls that guarantee the forward invariance of  $B_{\geq 0}$ . That means that all the trajectories that start in  $B_{\geq 0}$  and follow control signals that satisfy (2) at all time instants, remain within it at all times, as stated in the following theorem. If the set of unsafe states is disjoint from  $B_{\geq 0}$  and the system starts from states in  $B_{\geq 0}$ , then the system can be kept safe by following such controls.

**Theorem 1** ([4]). *Any Lipschitz continuous control policy  $\pi : \mathcal{D} \rightarrow \mathcal{U}$  satisfying*

$$\forall x \in \mathcal{D}, \pi(x) \in K_{cbf} := \{u \in \mathcal{U} \mid \nabla B(x)(f(x) + g(x)u) + \gamma(B(x)) \geq 0\}$$

*makes the set  $B_{\geq 0}$  forward invariant.*

Given a reference controller  $\pi_{\text{ref}} : \mathcal{X} \rightarrow \mathcal{U}$  that does not necessarily satisfy safety constraints, a safety filter using the CBF  $B$  can be employed to modify its unsafe control choices. Specifically, a quadratic program (QP) can be formulated to determine the closest safe control to  $\pi_{\text{ref}}(x)$  [5], resulting in what is usually denoted by a CBF-QP policy  $\pi_{\text{safe}}$ , as shown below:

$$\pi_{\text{safe}}(x) := \arg \min_{u \in \mathcal{U}} \|u - \pi_{\text{ref}}(x)\|^2 \quad \text{s.t.} \quad \nabla B(x)(f(x) + g(x)u) + \gamma(B(x)) \geq 0. \quad (3)$$

### 3.2 Offline reinforcement learning

Offline reinforcement learning enables designing optimal control policies from offline datasets that avoid OOD states and actions. CQL [23] is an offline RL algorithm that penalizes the  $Q$ -values of state-action pairs that do not appear in the offline dataset to prevent over-estimating their values.

The standard Q-learning objective based on the Bellman equation is to minimize the following loss function:  $\mathcal{L}_{\text{base}}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} [(Q_{\theta}(s,a) - (r + \gamma \max_{a'} Q_{\theta}(s',a')))^2]$ , where  $\mathcal{D}$  is the offline dataset,  $Q_{\theta}$  is the  $Q$  function being learned with parameters  $\theta$ , and  $\gamma$  is the discount factor [49]. CQL augments this function with another term that aims to prevent over-estimating the  $Q$ -values of OOD data as follows:  $\mathcal{L}_{\text{CQL}}(\theta) = \mathcal{L}_{\text{base}}(\theta) + \alpha \cdot \mathcal{L}_{\text{cons}}(\theta)$ , where  $\alpha$  is a hyperparameter and  $\mathcal{L}_{\text{cons}}$  is a regularization term that penalizes the  $Q$ -values of OOD state-action pairs. In practice,  $\mathcal{L}_{\text{cons}}$  can be implemented in various ways, one of which is the following:  $\mathcal{L}_{\text{cons}}(\theta) = \mathbb{E}_{s \sim \mathcal{D}} [\log \sum_a \exp(Q_{\theta}(s,a)) - \mathbb{E}_{a \sim \hat{\pi}_{\beta}(\cdot|s)} [Q_{\theta}(s,a)]]$ , where  $\hat{\pi}_{\beta}(a|s)$  is the *empirical behavior policy*, i.e., an estimate of the conditional distribution of actions given state  $s$  induced by the offline dataset  $\mathcal{D}$ . The first part of the term,  $\log \sum_a \exp(Q_{\theta}(s,a))$ , is a soft maximum over the  $Q$ -values corresponding to all possible actions at a given state. Minimizing this term reduces the  $Q$ -values of all actions. The second component,  $\mathbb{E}_{a \sim \hat{\pi}_{\beta}(\cdot|s)} [Q_{\theta}(s,a)]$ , is the expected  $Q$ -value for actions followed at state  $s$  in the trajectories in the dataset. Maximizing this term encourages assigning high  $Q$ -values for InD actions. As a result, unseen actions receive lower  $Q$ -values while in-distribution actions retain accurate estimates, so the derived policy  $\pi(s) = \arg \max_a Q(s,a)$  would select in-distribution actions.

### 3.3 Offline safe reinforcement learning

Safe RL extends RL to settings where an agent has to keep the total cost below a predefined threshold while maximizing reward. It aims to train a policy  $\pi$  for a Constrained Markov Decision Process (CMDP) that maximizes  $\mathbb{E}_s[V_r^{\pi}(s)]$  subject to the constraints  $\mathbb{E}_s[V_c^{\pi}(s)] \leq \kappa$  and  $D(\pi \parallel \pi_{\beta}) \leq \varepsilon$ , where  $V_r^{\pi}(s)$  represents the reward value function,  $V_c^{\pi}(s)$  represents the cost value function,  $\kappa$  is a user-defined cost threshold,  $\pi_{\beta}$  is the potentially-unknown behavioral policy that was used to generate the offline dataset  $\mathcal{D} := \{(s,a,s',r,c)\}$ , and  $D(\pi \parallel \pi_{\beta})$  measures the divergence between  $\pi_{\beta}$  and  $\pi$ .

### 3.4 Problem statement

We consider the setting where an offline dataset consisting of safe and unsafe trajectories is available. Our objective is to train a safety filter that prevents the system from reaching unsafe states. Consequently, it must also prevent the system from reaching OOD ones at which it might not be reliable enough to determine safe from unsafe states and actions. The safety filter should not be conservative beyond what is necessary to maintain safety, i.e., as minimally affecting nominal control as possible.

## 4 Method

Our aim is to design a CBF-QP policy from offline data. If the reference policy or the dynamics are not available, we train corresponding models using the same data. Moreover, in settings with

high-dimensional observations, we train representation models and consider the latent space to be the state space based on which we train the dynamics model and the neural CBF. We discuss how we train each of these components next.

#### 4.1 Learning latent dynamics

We train an autoencoder to project high-dimensional visual observations into a low-dimensional latent space. We train it in parallel with a control-affine latent dynamics model. The deterministic autoencoder maps RGB images  $o_t \in \mathbb{R}^{c \times h \times w}$  to latent vectors  $x_t \in \mathcal{X}$  using an encoder  $\mathcal{E}$ , and reconstructs them using a decoder  $\mathcal{G}$ . We train a control-affine dynamics model of the form:

$$x_{t+1} = x_t + (f_\theta(x_t) + g_\theta(x_t)u_t)\Delta t, \quad (4)$$

where  $f_\theta$  and  $g_\theta$  are multilayer perceptrons parametrized by  $\theta$ , approximating the continuous-time dynamics model  $\dot{x} = f_\theta(x) + g_\theta(x)u$ . We train the system by minimizing the following loss function using gradient descent:  $\mathcal{L} := \lambda_1 \|o_t - \mathcal{G}(\mathcal{E}(o_t))\|_2^2 + \lambda_2 \|\mathcal{E}(o_{t+1}) - \hat{x}_{t+1}\|_2^2 + \lambda_3 \|o_{t+1} - \mathcal{G}(\hat{x}_{t+1})\|_2^2$ , where  $\hat{x}_{t+1}$  is the predicted next latent state using (4), and  $\lambda_1$ ,  $\lambda_2$ , and  $\lambda_3$  are hyperparameters. The first term ensures that the latent vector retains sufficient information for accurate image reconstruction, the second enforces accurate latent dynamics prediction, and the third maintains consistency between predicted and ground-truth observations. During training, we optimize the autoencoder's parameters using the first loss term  $\lambda_1 \|o_t - \mathcal{G}(\mathcal{E}(o_t))\|_2^2$ , while holding those parameters fixed when optimizing the dynamics model to maintain a stable latent state space.

#### 4.2 Learning nominal controllers

We train neural reference controllers from offline data using behavior cloning (BC) and safe offline RL algorithms. We train BC-based controllers by solving the optimization problem  $\operatorname{argmax}_\beta \mathbb{E}_{(x,u) \sim \mathcal{D}} [\log \pi_\beta(u|x)]$ . When we use only the safe trajectories in the dataset for behavior cloning, we call the resulting policy BC-Safe. For safety gymnasium environments [50], we train BC and safe offline-RL policies using DSRL [51], which provides training datasets and reference implementations for safe offline-RL algorithms.

#### 4.3 Learning conservative control barrier functions from offline datasets

We assume that a dataset of trajectories of the form  $\sigma := \{x_1, a_1, x_2, a_2, \dots, x_T, a_T\}$  is available. We define  $\mathcal{X}_{\text{safe}}$  as the set of safe states and  $\mathcal{X}_{\text{unsafe}}$  as the set of unsafe states in the dataset. We denote the set of consecutive triplets  $(x, a, x')$  in the dataset in which  $x' \in \mathcal{X}_{\text{safe}}$  by  $\mathcal{D}_{\text{safe}}$  and the set of triplets in which  $x' \in \mathcal{X}_{\text{unsafe}}$  by  $\mathcal{D}_{\text{unsafe}}$ . We also define  $\mathcal{D} := \mathcal{D}_{\text{safe}} \cup \mathcal{D}_{\text{unsafe}}$ . Given this dataset, we train a neural CCBF  $B_\phi : \mathcal{X} \rightarrow \mathbb{R}$  by minimizing the following loss function:

$$\mathcal{L}_{\text{CCBF}}(\phi) := \mathcal{L}_{\text{safe}}(\phi) + \mathcal{L}_{\text{unsafe}}(\phi) + \mathcal{L}_{\text{ascent}}(\phi) + \mathcal{L}_{\text{descent}}(\phi) + \mathcal{L}_{\text{lip}}(\phi) + \mathcal{L}_c(\phi), \quad (5)$$

where

$$\begin{aligned} \mathcal{L}_{\text{safe}}(\phi) &:= \frac{w_{\text{safe}}}{|\mathcal{X}_{\text{safe}}|} \sum_{x \in \mathcal{X}_{\text{safe}}} \sigma(\epsilon_{\text{safe}} - B_\phi(x)), \quad \mathcal{L}_{\text{unsafe}}(\phi) := \frac{w_{\text{unsafe}}}{|\mathcal{X}_{\text{unsafe}}|} \sum_{x \in \mathcal{X}_{\text{unsafe}}} \sigma(\epsilon_{\text{unsafe}} + B_\phi(x)), \\ \mathcal{L}_{\text{ascent}}(\phi) &:= \frac{w_{\text{ascent}}}{|\mathcal{D}_{\text{safe}}|} \sum_{(x,u,x') \in \mathcal{D}_{\text{safe}}} \sigma(\epsilon_{\text{ascent}} - \nabla B_\phi(x) \cdot (f_\theta(x) + g_\theta(x)u) - \gamma B_\phi(x)), \\ \mathcal{L}_{\text{descent}}(\phi) &:= \frac{w_{\text{descent}}}{|\mathcal{D}_{\text{unsafe}}|} \sum_{(x,u,x') \in \mathcal{D}_{\text{unsafe}}} \sigma(\epsilon_{\text{descent}} + \nabla B_\phi(x) \cdot (f_\theta(x) + g_\theta(x)u_{\text{unsafe}}) + \gamma B_\phi(x)), \\ \mathcal{L}_{\text{lip}}(\phi) &:= \frac{w_{\text{lip}}}{|\mathcal{D}|} \sum_{(x,u,x') \in \mathcal{D}} \|B_\phi(x') - B_\phi(x)\|, \text{ and } \mathcal{L}_c(\phi) := \frac{w_c}{|\mathcal{X}_{\text{safe}}|} \sum_{x \in \mathcal{X}_{\text{safe}}} \left[ \tau \ln \left( \sum_{v \in V_x} \exp \left( \frac{B_\phi(x'_v)}{\tau} \right) \right) \right]. \end{aligned}$$

Here,  $\sigma$  is the ReLU activation function, and  $\epsilon_{\text{safe}}$ ,  $\epsilon_{\text{unsafe}}$ ,  $\epsilon_{\text{ascent}}$ ,  $\epsilon_{\text{descent}}$ ,  $w_{\text{safe}}$ ,  $w_{\text{unsafe}}$ ,  $w_{\text{ascent}}$ ,  $w_{\text{descent}}$ ,  $w_{\text{lip}}$ ,  $w_c$ , and  $\tau$  are hyperparameters. The set  $V_x$  is a uniformly sampled set of actions from  $\mathcal{U}$  to evaluate at state  $x$ . The state  $x'_v$  is the state reached by the learned dynamics following action  $v$  at



state  $x$ . The state  $x'$  is the next state following  $x$  in the dataset, i.e., the one reached by system (1) following action  $u$  at state  $x$ , where  $u$  is the second element in the triple  $(x, u, x') \in \mathcal{D}$ . When we train without using  $\mathcal{L}_c$ , we refer to the result as a neural control barrier function (NCBF).

The loss terms  $\mathcal{L}_{\text{safe}}$  and  $\mathcal{L}_{\text{unsafe}}$  drive the CCBF to be positive at safe states and negative at unsafe states. The terms  $\mathcal{L}_{\text{ascent}}$  and  $\mathcal{L}_{\text{descent}}$  drive the CCBF to satisfy the gradient constraint (2). Moreover, the loss term  $\mathcal{L}_{\text{lip}}$  drives the learned CCBF to be Lipschitz continuous. Previous works on learning neural CBFs from data [52, 37, 11] have used only  $\mathcal{L}_{\text{safe}}$ ,  $\mathcal{L}_{\text{unsafe}}$  and  $\mathcal{L}_{\text{ascent}}$  in their loss computation. We added the new term  $\mathcal{L}_{\text{lip}}$  to smoothen the learned neural CBF. Finally,  $\mathcal{L}_c$  drives the CCBF to be conservative in its values on OOD states.

The term inside the sum in  $\mathcal{L}_c$  represents the soft maximum of the CCBF values of the sampled states as well as the following state appearing in the training trajectory. The temperature parameter  $\tau$  controls the smoothness of the soft maximum. As  $\tau$  decreases, the soft maximum approaches the actual maximum. The gradient of  $\mathcal{L}_c$  is equal to the gradient of  $\frac{w_c}{|\mathcal{X}_{\text{safe}}|} \sum_{x \in \mathcal{X}_{\text{safe}}} \left[ \tau \ln \left( \sum_{v \in V_x} \exp \left( \frac{B_\phi(x'_v)}{\tau} \right) \right) - B_\phi(x') \right]$ , but with the second term,  $B_\phi(x')$ , detached from the computational graph during training in Pytorch using the `tensor.detach()` method. The latter is similar to the loss used in conservative Q learning (CQL) [23], where the Q-function is replaced by  $B_\phi$ , but CQL does not detach the second term. The reason we do so is to minimize  $\mathcal{L}_c$  by decreasing the values in  $\{B_\phi(x'_v)\}_{v \in V_x}$  instead of increasing the value of  $B_\phi(x')$ . We want the CCBF values at InD states to be determined mainly by the loss terms  $\mathcal{L}_{\text{safe}}$ ,  $\mathcal{L}_{\text{unsafe}}$ ,  $\mathcal{L}_{\text{ascent}}$ ,  $\mathcal{L}_{\text{descent}}$ , and  $\mathcal{L}_{\text{lip}}$ , which enforce the conditions required by the learned CBF to be a valid CBF, while minimally being affected by the conservative loss term  $\mathcal{L}_c$ . We did an ablation study where we did not detach  $B_\phi(x')$ . Its results, shown in Appendix E, support our argument that the learned CCBFs without detaching  $B_\phi(x')$  generally under-perform those obtained by detaching it. Results of the latter are shown in Section 5.2.

## 5 Case Studies

In this section, we describe the experimental setup across four environments and subsequently present our results along with the key observations.

### 5.1 Experimental setup

We evaluate our approach on four environments: (1) **2D navigation**: an agent must navigate to a goal position while avoiding a static circular obstacle, (2) **Vision-based navigation**: similar to 2D navigation but using bird-eye-view image observations, (3) **SafetyHopper** and (4) **SafetySwimmer**: locomotion tasks from the safety gymnasium benchmark, where agents must move as quickly as possible while adhering to velocity constraints. More details on these environments can be found in Appendix A. In each experiment, we present the results of rolling out the policy under different conditions: with no safety filters and with a NCBF-, an iDBF- and a CCBF-based ones. We compare these methods with FISOR [46] in the safety gymnasium tasks [50]. We train the NCBF, iDBF, and CCBF using the same hyperparameters including model size and only modify the hyperparameters related to the iDBF- and CCBF-specific loss terms.

**2D navigation** The agent has 2D single-integrator dynamics, has actuation limits that constrain the maximum speed in each dimension to be  $v_{\text{max}}$ , and has to reach a goal position while avoiding a static circular obstacle. A state is considered safe if the Euclidean distance  $d$  to the obstacle is greater than or equal to  $r + \epsilon$ , where  $r$  is the obstacle’s radius and  $\epsilon = 3$ . States with  $d \leq r$  are labeled unsafe, and states in between are not used for training.

**Vision-based navigation** This setup extends the 2D navigation task to a vision-based setting, where  $64 \times 64 \times 3$  RGB observations representing a top-down view of the environment are given instead of the state, i.e., the 2D position.

**Safety gymnasium locomotion** We test on SafetyHopper (11D state, 3D control) and SafetySwimmer (8D state, 2D control) from safety gymnasium [50]. Agents earn rewards for forward motion but incur costs if velocity exceeds user-defined thresholds. States are labeled unsafe if they exceed the maximum velocity constraint, and safe otherwise.

## 5.2 Results

We discuss two main observations: (1) CCBF-based filters significantly improves safety while minimally affecting performance, and (2) CCBF is easier to train and more robust to hyperparameters change than baselines. Further observations are discussed in Appendix C. We report the results for the 2D navigation and vision-based navigation scenarios in Table 1 using the metrics: **success %**, the percentage of episodes in which the agent successfully reached the goal; and **collision %**, the percentage of episodes with at least one collision. We visualize some of the NCBFs learned in the 2D navigation task in Figure 1 and provide additional visualizations in Figures 4a and 4b (in Appendix D). We report our results for the safety gymnasium tasks in Table 2 using the metrics: **normalized reward** and **normalized cost**, following the steps of previous works [51, 53, 54]. Detailed metric definitions can be found in Appendix A.4.2. Finally, Appendix B presents ablation studies investigating the effect of varying the density threshold  $p$  for training the iDBFs as well as varying model sizes.

Table 1: Evaluation results for different pairs of nominal controllers and safety filters. Results of each pair are averaged across 500 runs for the 2D navigation task and 50 runs for the vision-based navigation task. **Green**: Safest agent with the highest average success % across all controllers.

Policy	None		NCBF		CCBF		iDBF	
	Success %	Collision %	Success %	Collision %	Success %	Collision %	Success %	Collision %
<i>2D Navigation</i>								
PD controller	–	–	91.6±2.7	0	91.0±2.2	0	0	100.0
BC	96.2±2.2	21.8±4.2	95.2±2.3	0	95.4±2.6	0	9.6±2.1	3.4±1.5
BC-Safe	93.6±3.2	6.4±2.3	93.8±1.9	0	92.8±3.6	0	7.8±0.8	0
<b>Average</b>	94.9	14.1	<b>93.5</b>	<b>0</b>	93.1	0	5.8	34.5
<i>Vision-based Navigation</i>								
PD controller	–	–	34.0±15.2	20.0±7.1	58.0±14.8	4.0±5.5	94.0±8.9	80.0±15.8
BC	98.0±4.5	36.0±16.7	68.0±19.2	6.0±5.5	86.0±11.4	0	96.0±5.5	14.0±16.7
BC-Safe	92.0±8.4	22.0±4.5	78.0±16.4	16.0±8.9	94.0±8.9	0	98.0±4.5	10.0±10.0
<b>Average</b>	95.0	29.0	60.0	14.0	<b>79.3</b>	<b>1.3</b>	96.0	34.7

Table 2: Evaluation results of normalized reward ( $R\uparrow$ ) and normalized cost ( $C\downarrow$ ) for different pairs of nominal controllers and safety filters. Results are averaged across three barrier models trained with each method and each evaluated on 20 runs starting from random initial states. The  $\uparrow$  symbol denotes that the higher reward, the better. The  $\downarrow$  symbol denotes that the lower cost, the better. **Bold**: Safe agents ( $\text{Cost} < 1$ ). **Gray**: Unsafe agents ( $\text{Cost} \geq 1$ ). **Green**: Safest agent with the highest average success % across all controllers.

Policy	None		NCBF		CCBF		iDBF	
	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$
<i>Swimmer</i>								
BC	2.26±0.64	0.44±0.03	5.05±2.84	0.4±0.06	<b>0.94±0.18</b>	<b>0.39±0.03</b>	3.80±0.31	0.36±0.01
BC-Safe	<b>0.12±0.05</b>	<b>0.43±0.03</b>	<b>0.12±0.06</b>	<b>0.46±0.02</b>	<b>0.03±0.02</b>	<b>0.45±0.02</b>	<b>0.28±0.09</b>	<b>0.50±0.02</b>
COptiDICE	1.65±0.33	0.30±0.04	1.40±0.39	0.07±0.01	<b>0.19±0.08</b>	<b>0.28±0.06</b>	11.75±2.99	0.57±0.03
BEAR-L	<b>0.61±0.11</b>	<b>0.16±0.03</b>	<b>0.46±0.05</b>	<b>0.18±0.05</b>	<b>0.42±0.17</b>	<b>0.09±0.01</b>	<b>0.43±0.07</b>	<b>0.14±0.01</b>
BCQ-L	2.82±1.15	0.25±0.08	4.69±3.20	0.20±0.10	1.40±0.47	0.29±0.03	9.38±1.65	0.39±0.08
<b>Average</b>	1.49	0.32	2.34	0.26	<b>0.59</b>	<b>0.3</b>	5.1	0.39
FISOR	<b>0.01±0.01</b>	<b>-0.08±0.01</b>	-	-	-	-	-	-
<i>Hopper</i>								
BC	<b>0.19±0.27</b>	<b>0.04±0.02</b>	<b>0.23±0.18</b>	<b>0.04±0.01</b>	<b>0.09±0.06</b>	<b>0.04±0.01</b>	4.71±2.26	0.33±0.24
BC-Safe	<b>0.03±0.02</b>	<b>0.57±0.01</b>	<b>0.14±0.10</b>	<b>0.61±0.01</b>	<b>0.05±0.03</b>	<b>0.56±0.05</b>	<b>0.48±0.22</b>	<b>0.56±0.03</b>
COptiDICE	<b>0.01±0.01</b>	<b>0.18±0.01</b>	<b>0.01±0.01</b>	<b>0.17±0.01</b>	<b>0.03±0.03</b>	<b>0.17±0.01</b>	1.81±0.73	0.24±0.03
BEAR-L	<b>0.37±0.01</b>	<b>0.16±0.01</b>	<b>0.27±0.09</b>	<b>0.21±0.08</b>	<b>0.20±0.06</b>	<b>0.30±0.06</b>	<b>0.34±0.02</b>	<b>0.16±0.01</b>
BCQ-L	3.09±0.34	0.50±0.04	1.34±0.48	0.31±0.02	1.16±0.53	0.37±0.05	3.82±0.17	0.46±0.20
<b>Average</b>	<b>0.74</b>	<b>0.29</b>	<b>0.39</b>	<b>0.27</b>	<b>0.31</b>	<b>0.28</b>	2.23	0.35
FISOR	<b>0.03±0.04</b>	<b>0.08±0.05</b>	-	-	-	-	-	-

**CCBF-based filters improve safety significantly while minimally affecting rewards** Observing the results in Table 1 for the 2D navigation task, we can see that NCBF and CCBF result in similar

success percentages and both result in zero collisions, outperforming iDBF. In the vision-based navigation environment, CCBF outperforms both NCBF and iDBF by achieving better success and collision percentages using any nominal controller. Unlike the case of the 2D navigation scenario, iDBF generally performs better than NCBF. The improved performance of iDBF can be attributed to the naturally increased distance between training trajectories, which reduces the likelihood of mislabeling nearby images as safe or unsafe as the sampled images nearby a training trajectory will not overlap with the images visited by other training trajectories. In contrast, the training trajectories in the 2D environment are more closely spaced, resulting in the sampled states by iDBF, which are labeled as unsafe, overlapping with the states of training trajectories which are labeled as safe.

In the Swimmer environment, CCBF reduces the average cost over all nominal controllers by approximately 60% with around a 6% drop in average reward. In contrast, both iDBF and NCBF increase the cost. In the Hopper environment, both CCBF and iDBF exhibit similar behavior seen in the Swimmer environment, whereas NCBF reduces the average cost by 47% with only a 7% drop in average reward, which differs from its performance in the Swimmer environment. These results show that CCBF consistently reduces cost while minimally impacting reward. On the other hand, NCBF’s results vary between environments—performing poorly in Swimmer but well in Hopper—possibly because Swimmer results in a larger distribution shift during deployment.

That said, FISOR results in the lowest cost in the Swimmer environment. It also almost results in the lowest cost in the Hopper environment—comparable to BC-safe and only slightly less safe than COptiDICE. However, it underperforms in terms of reward achieved in both environments. In the Swimmer environment, it results in a reward of -0.08, whereas CCBF paired with BC-Safe results in a 0.45 reward. Similarly, in the Hopper environment, its reward return is 0.08 compared to 0.17 resulting from CCBF paired with the COptiDICE-based nominal controller.

**CCBF is easier to train and more robust to hyperparameters** In all experiments, CCBFs consistently distinguished safe and unsafe states when using different values for  $w_c$  in the loss term  $\mathcal{L}_c$ . Figure 4a (in Appendix D) visualizes the CCBFs learned using different values of  $w_c$  for the 2D navigation example. All of them classify the states in the obstacle as unsafe and the states visited by the safe trajectories as safe. However, tuning  $w_c$  and  $\tau$  is needed for CCBF to better classify OOD states as unsafe. On the other hand, iDBF’s performance seems to be more sensitive to the density threshold  $p$  that is used to classify sampled actions as OOD. It is often the case that a choice of  $p$ , especially when it is large (e.g.,  $p = 0.1$ ), requires the tuning of the other hyperparameters  $w_{\text{safe}}$ ,  $w_{\text{unsafe}}$ ,  $\epsilon_{\text{safe}}$ , and  $\epsilon_{\text{unsafe}}$  to prevent training an iDBF that assigns a single value to all states. As can be seen in Figure 4b (in Appendix D), without the right choice of  $p$ , iDBF does not properly classify the safe, unsafe, InD, and OOD states. We observed similar behavior of iDBF in the vision-based navigation and safety gymnasium scenarios.

## 6 Conclusions and Limitations

We introduced Conservative Control Barrier Functions (CCBFs), neural safety filters trained using offline data. The idea is to add a loss term that penalizes over-optimistic safety estimations of OOD states to the loss function for training neural CBFs, resulting in safety estimations that better differentiate safe, unsafe, and InD and OOD states. When used to define quadratic programs as safety filters, they result in closed-loop behaviors that avoid unsafe and OOD states. We evaluated our approach in multiple offline environments and showed that it outperforms existing baselines in maintaining safety while minimally affecting performance.

Our approach in its present form does not offer formal guarantees that it will maintain safety or OOD states avoidance, and deriving such guarantees would be an important future direction. Moreover, evaluating generated safety filters in more realistic environments including physical robots operating in real environments would be equally valuable. Also, using world models [55, 56] to model the dynamics over the latent space would be needed to model more complex environments that are partially-observed. Finally, using dynamic programming-like approaches for training the safety filters might be needed when the states in the offline datasets are not labeled as safe and unsafe, but functions that define failure sets are available [21, 57].



## References

- [1] J. Betz, A. Heilmeier, A. Wischnewski, T. Stahl, and M. Lienkamp. Autonomous driving—a crash explained in detail. *Applied Sciences*, 9(23), 2019. ISSN 2076-3417. doi:10.3390/app9235126. URL <https://www.mdpi.com/2076-3417/9/23/5126>.
- [2] T. Haidegger. Autonomy for surgical robots: Concepts and paradigms. *IEEE Transactions on Medical Robotics and Bionics*, 1(2):65–76, 2019. doi:10.1109/TMRB.2019.2913282.
- [3] K.-C. Hsu, H. Hu, and J. F. Fisac. The safety filter: A unified view of safety-critical control in autonomous systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 7, 2023.
- [4] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pages 3420–3431, 2019. doi:10.23919/ECC.2019.8796030.
- [5] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada. Control barrier function based quadratic programs for safety critical systems. *IEEE Transactions on Automatic Control*, 62(8):3861–3876, 2016.
- [6] A. A. Ahmadi and A. Majumdar. Some applications of polynomial optimization in operations research and real-time decision making. *Optimization Letters*, 10(4):709–729, Apr. 2016. ISSN 1862-4480. doi:10.1007/s11590-015-0894-3. URL <https://doi.org/10.1007/s11590-015-0894-3>.
- [7] A. Clark. Verification and synthesis of control barrier functions. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 6105–6112. IEEE, 2021.
- [8] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni. Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3717–3724. Ieee, 2020.
- [9] K. P. Wabersich, A. J. Taylor, J. J. Choi, K. Sreenath, C. J. Tomlin, A. D. Ames, and M. N. Zeilinger. Data-driven safety filters: Hamilton-jacobi reachability, control barrier functions, and predictive methods for uncertain systems. *IEEE Control Systems Magazine*, 43(5):137–177, 2023.
- [10] C. Dawson, S. Gao, and C. Fan. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, 39(3):1749–1767, 2023. doi:10.1109/TRO.2022.3232542.
- [11] C. Dawson, Z. Qin, S. Gao, and C. Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. In *Conference on Robot Learning*, pages 1724–1735. PMLR, 2022.
- [12] L. Lindemann, A. Robey, L. Jiang, S. Das, S. Tu, and N. Matni. Learning robust output control barrier functions from safe expert demonstrations. *IEEE Open Journal of Control Systems*, 2024.
- [13] H. Abdi, G. Raja, and R. Ghabcheloo. Safe control using vision-based control barrier function (v-cbf). In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 782–788. IEEE, 2023.
- [14] M. Tong, C. Dawson, and C. Fan. Enforcing safety for vision-based controllers via control barrier functions and neural radiance fields. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10511–10517. IEEE, 2023.
- [15] W. Xiao, T.-H. Wang, M. Chahine, A. Amini, R. Hasani, and D. Rus. Differentiable control barrier functions for vision-based end-to-end autonomous driving. *arXiv preprint arXiv:2203.02401*, 2022.

- [16] Y. Yang and H. Sibai. Pre-trained vision models as perception backbones for safety filters in autonomous driving, 2024. URL <https://arxiv.org/abs/2410.22585>.
- [17] I. Tabbara and H. Sibai. Learning Ensembles of Vision-based Safety Control Filters. *arXiv e-prints*, art. arXiv:2412.02029, Dec. 2024. doi:10.48550/arXiv.2412.02029.
- [18] K. Nakamura, L. Peters, and A. Bajcsy. Generalizing safety beyond collision-avoidance via latent-space reachability analysis. *arXiv preprint arXiv:2502.00935*, 2025.
- [19] T. He, C. Zhang, W. Xiao, G. He, C. Liu, and G. Shi. Agile but safe: Learning collision-free high-speed legged locomotion. *arXiv preprint arXiv:2401.17583*, 2024.
- [20] S. Keyumarsi, M. W. S. Atman, and A. Gusrialdi. Lidar-based online control barrier function synthesis for safe navigation in unknown environments. *IEEE Robotics and Automation Letters*, 9(2):1043–1050, 2023.
- [21] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11532–11539, 2024. doi:10.1109/ICRA57147.2024.10610418.
- [22] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- [23] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.
- [24] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=68n2s9ZJWF8>.
- [25] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.
- [26] Y. Wu, S. Zhai, N. Srivastava, J. Susskind, J. Zhang, R. Salakhutdinov, and H. Goh. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*, 2021.
- [27] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in neural information processing systems*, 32, 2019.
- [28] J. Lee, C. Paduraru, D. J. Mankowitz, N. Heess, D. Precup, K.-E. Kim, and A. Guez. Coptidice: Offline constrained reinforcement learning via stationary distribution correction estimation. *arXiv preprint arXiv:2204.08957*, 2022.
- [29] G. Chou, N. Ozay, and D. Berenson. Learning constraints from locally-optimal demonstrations under cost function uncertainty. *IEEE Robotics and Automation Letters*, 5(2):3682–3690, 2020.
- [30] K. Kim, G. Swamy, Z. Liu, D. Zhao, S. Choudhury, and S. Z. Wu. Learning shared safety constraints from multi-task demonstrations. *Advances in Neural Information Processing Systems*, 36:5808–5826, 2023.
- [31] K. Leung, S. Veer, E. Schmerling, and M. Pavone. Learning autonomous vehicle safety concepts from demonstrations. In *2023 American Control Conference (ACC)*, pages 3193–3200. IEEE, 2023.
- [32] D. Lindner, X. Chen, S. Tschitschek, K. Hofmann, and A. Krause. Learning safety constraints from demonstrations with unknown rewards. In *International Conference on Artificial Intelligence and Statistics*, pages 2386–2394. PMLR, 2024.

- [33] D. R. Scobee and S. S. Sastry. Maximum likelihood constraint inference for inverse reinforcement learning. *arXiv preprint arXiv:1909.05477*, 2019.
- [34] H. Yu, S. Farrell, R. Yoshimitsu, Z. Qin, H. I. Christensen, and S. Gao. Estimating control barriers from offline data. *arXiv preprint arXiv:2503.10641*, 2025.
- [35] M. Qadri, G. Swamy, J. Francis, M. Kaess, and A. Bajcsy. Your learned constraint is secretly a backward reachable tube. *arXiv preprint arXiv:2501.15618*, 2025.
- [36] O. So, Z. Serlin, M. Mann, J. Gonzales, K. Rutledge, N. Roy, and C. Fan. How to train your neural control barrier function: Learning safety filters for complex input-constrained systems. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11532–11539. IEEE, 2024.
- [37] F. Castaneda, H. Nishimura, R. T. McAllister, K. Sreenath, and A. Gaidon. In-distribution barrier functions: Self-supervised policy filters that avoid out-of-distribution states. In *Learning for Dynamics and Control Conference*, pages 286–299. PMLR, 2023.
- [38] K. Kang, P. Gradu, J. J. Choi, M. Janner, C. Tomlin, and S. Levine. Lyapunov density models: Constraining distribution shift in learning-based control. In *International Conference on Machine Learning*, pages 10708–10733. PMLR, 2022.
- [39] E. D. Sontag. Control-lyapunov functions. In *Open problems in mathematical systems and control theory*, pages 211–216. Springer, 1999.
- [40] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.
- [41] H. Yu, S. Farrell, R. Yoshimitsu, Z. Qin, H. I. Christensen, and S. Gao. Estimating control barriers from offline data. *arXiv preprint arXiv:2503.10641*, 2025.
- [42] N. Charoenphakdee, Z. Cui, Y. Zhang, and M. Sugiyama. Classification with rejection based on cost-sensitive classification. In *International Conference on Machine Learning*, pages 1507–1517. PMLR, 2021.
- [43] A. Robey, L. Lindemann, S. Tu, and N. Matni. Learning robust hybrid control barrier functions for uncertain systems. *IFAC-PapersOnLine*, 54(5):1–6, 2021.
- [44] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, and A. Knoll. A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*, 2022.
- [45] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.
- [46] Y. Zheng, J. Li, D. Yu, Y. Yang, S. E. Li, X. Zhan, and J. Liu. Safe offline reinforcement learning with feasibility-guided diffusion model. *arXiv preprint arXiv:2401.10700*, 2024.
- [47] S. Qin, Y. Yang, Y. Mu, J. Li, W. Zou, J. Duan, and S. E. Li. Feasible reachable policy iteration. In *Forty-first International Conference on Machine Learning*, 2024.
- [48] S. Zhao, J. Zhang, N. Masoud, J. Li, Y. Zheng, and X. Hou. Reachability-aware reinforcement learning for collision avoidance in human-machine shared control. *arXiv preprint arXiv:2502.10610*, 2025.
- [49] J. Clifton and E. Laber. Q-learning: Theory and applications. *Annual Review of Statistics and Its Application*, 7(1):279–301, 2020.
- [50] J. Ji, B. Zhang, J. Zhou, X. Pan, W. Huang, R. Sun, Y. Geng, Y. Zhong, J. Dai, and Y. Yang. Safety gymnasium: A unified safe reinforcement learning benchmark. *Advances in Neural Information Processing Systems*, 36:18964–18993, 2023.

- [51] Z. Liu, Z. Guo, H. Lin, Y. Yao, J. Zhu, Z. Cen, H. Hu, W. Yu, T. Zhang, J. Tan, et al. Datasets and benchmarks for offline safe reinforcement learning. *arXiv preprint arXiv:2306.09303*, 2023.
- [52] Y. Qin, N. Mote, H. Nishimura, and A. D. Ames. Sablas: Learning safe control barrier functions for systems with unknown dynamics. *IEEE Robotics and Automation Letters*, 7(3):7357–7364, 2022.
- [53] Z. Liu, Z. Guo, Y. Yao, Z. Cen, W. Yu, T. Zhang, and D. Zhao. Constrained decision transformer for offline safe reinforcement learning. In *International Conference on Machine Learning*, pages 21611–21630. PMLR, 2023.
- [54] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.
- [55] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models, 2023. URL <https://arxiv.org/abs/2301.04104>, 2023.
- [56] G. Zhou, H. Pan, Y. LeCun, and L. Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.
- [57] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.
- [58] H. Xu, X. Zhan, and X. Zhu. Constraints penalized q-learning for safe offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8753–8760, 2022.
- [59] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pages 2052–2062. PMLR, 2019.
- [60] J. Li, X. Zhan, H. Xu, X. Zhu, J. Liu, and Y.-Q. Zhang. When data geometry meets deep function: Generalizing offline reinforcement learning. *arXiv preprint arXiv:2205.11027*, 2022.

## A Additional details on experimental setup

### A.1 Environments

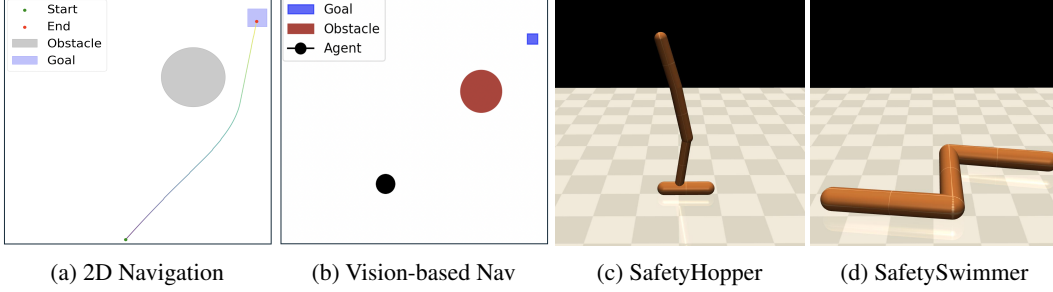


Figure 2: Illustration of the four experimental environments used in our evaluation.

### A.2 Experimental details for 2D navigation environment

#### A.2.1 Dataset generation and implementation details

We generated 2,000 diverse trajectories using a CBF-QP policy that combines a nominal PD controller with a manually-designed CBF. The nominal controller is defined as  $\mathbf{u}_{\text{nom}} = K_p(\mathbf{x}_g - \mathbf{x}) + K_d \frac{\mathbf{u}_{\text{nom}}}{\Delta t}$ , where  $\mathbf{x}_g$  is the goal position,  $\Delta t = 0.1$  seconds is the timestep (10 Hz frequency), and  $K_p, K_d$  are the proportional and derivative gains, respectively. The constraints in the QP are  $\dot{h}(\mathbf{x}) + \alpha h(\mathbf{x}) \geq 0$ , where  $h(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_{\text{obs}}\|^2 - r^2$  is the CBF, and  $-v_{\text{max}} \leq v_x \leq v_{\text{max}}$  and  $-v_{\text{max}} \leq v_y \leq v_{\text{max}}$ , where  $v_{\text{max}} = 3$ . Each trajectory is initialized from a random state. We gradually increase obstacle radius  $r$  as the dataset generation progresses. Initially, we set  $r$  to 0.01 and increment it by 0.2 up to 5, updating every 25000 time steps, where the time step count is cumulative over trajectories. The episode ends after 200 timesteps or when the agent reaches the goal, defined as  $\|\mathbf{x} - \mathbf{x}_g\|^2 < \delta$ , where  $\delta$  is 0.5. This setup generates a diverse dataset of safe and unsafe trajectories. We train the CCBF as described in Section 4.3, but we do not include the terms  $\mathcal{L}_{\text{descent}}$  and  $\mathcal{L}_{\text{lip}}$  in this scenario.

During evaluation, each episode starts from a randomly sampled initial state within the square  $[-18, 5]^2$ . We solve the QP without actuation constraints, and then scale the control input as  $\mathbf{u}_{\text{scaled}} = \frac{\mathbf{u}}{\|\mathbf{u}\|_2} v_{\text{max}}$  to ensure it satisfies the actuation constraints.

#### A.2.2 Visualization of a trajectory generated using CCBF

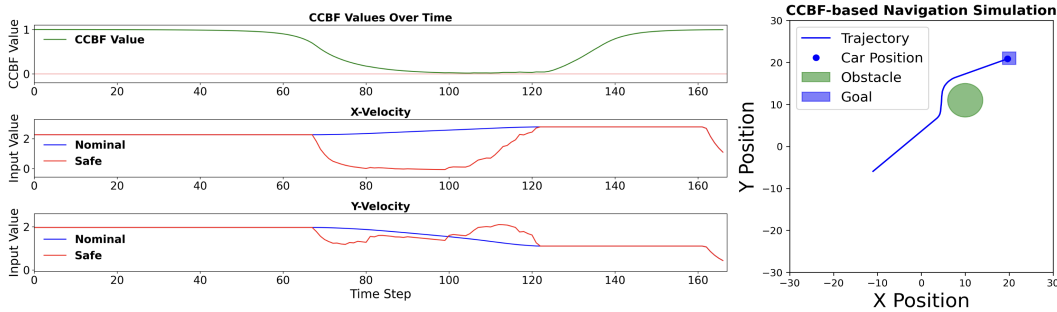


Figure 3: A trajectory generated using CCBF-QP in the 2D navigation task as well as the nominal and CCBF-QP controls.

### A.2.3 Hyperparameters

Table 3: Hyperparameters for NCBF, CCBF, iDBF in the 2D navigation task

Parameter	Value
Number of hidden layers	3
Hidden dimension	128
Batch size	128
Learning rate	$1 \times 10^{-4}$
$\epsilon_{\text{safe}}$	0.2
$\epsilon_{\text{unsafe}}$	0.2
$w_{\text{safe}}$	1.0
$w_{\text{unsafe}}$	1.2
$w_{\text{ascent}}$	1.0
$p$	$1 \times 10^{-8}$
$w_c$	0.5
$\tau$	0.7
Optimizer	Adam
Time step (dt)	0.1

## A.3 Experimental details for vision-based navigation environment

### A.3.1 Dataset generation and implementation details

Using the same setup as in the 2D navigation task, we set  $\delta = 2$  and collect 3,000 trajectories with  $64 \times 64 \times 3$  RGB image observations. They are a mix of safe and unsafe trajectories, generated by varying the obstacle radius and initial positions as before, with the maximum episode length being 20 seconds (200 timesteps, 10Hz controller). The collected image-based trajectories are then used to learn latent dynamics, as described in Section 4.1. An observation is considered safe if the Euclidean distance  $d$  to the obstacle is greater than or equal to  $r + \epsilon$ , where  $\epsilon = 3$ , otherwise it is unsafe. We train the CCBF as described in Section 4.3, but do not include  $\mathcal{L}_{\text{lip}}$ .

### A.3.2 Hyperparameters

Table 4: Hyperparameters for NCBF, CCBF, and iDBF in the vision-based navigation task

Parameter	Value
Latent dimension size	4
Action dimension size	2
Hidden dimension	128
Number of hidden layers	3
Batch size	256
Learning rate	$1 \times 10^{-4}$
$\epsilon_{\text{safe}}$	0.08
$\epsilon_{\text{unsafe}}$	0.15
$\epsilon_{\text{ascent}}$	0.02
$\epsilon_{\text{descent}}$	0.02
$w_{\text{ascent}}$	2.0
$w_{\text{descent}}$	1.0
$w_{\text{safe}}$	1.0
$w_{\text{unsafe}}$	1.1
$p$	$1 \times 10^{-6}$
$w_c$	1
$\tau$	0.7
Optimizer	Adam
Time step (dt)	0.1



Table 5: Hyperparameters for latent dynamics in the vision-based navigation task

Parameters	Value
Encoder CNN layers	Conv2d(3→32) Conv2d(32→64) Conv2d(64→128) Flatten Linear(8192→400) Linear(400→4) (latent state)
Decoder CNN layers	Linear(4→8192) Unflatten (128, 8, 8) ConvTranspose2d(128→64) ConvTranspose2d(64→32) ConvTranspose2d(32→3) Sigmoid activation (output $\in [0, 1]$ )
Input channels	3
Latent dimension size	4
Hidden dimension (Dynamics)	400
Number of hidden layers (Dynamics)	3
Learning rate (Dynamics)	$1 \times 10^{-4}$
Learning rate (encoder, decoder)	$1 \times 10^{-4}$
Batch size	32
$\lambda_1$	1.0
$\lambda_2$	1.0
$\lambda_3$	0.5
Optimizer	Adam
Time step (dt)	0.1

## A.4 Safety gymnasium Swimmer and Hopper environments

### A.4.1 Implementation and experimental details

We first train a dynamics model as described in Section 4.1, but instead of employing an autoencoder as in the vision-based task to map observations to latent states, we learn the dynamics directly in safety gymnasium’s native state space. Then, we train the CCBF as described in Section 4.3, but do not include  $\mathcal{L}_{\text{descent}}$ . We train the nominal controllers using the offline RL algorithms (COptiDICE [28], BEAR-L [58, 27], BCQ-L [58, 59]) and using BC variants as implemented in DSRL [51]. During evaluation, if the QP was infeasible, we select the closest point in  $\mathcal{U}$  to the constraint boundary.

### A.4.2 Normalized reward and normalized cost metrics in the safety gymnasium tasks

For both the Swimmer and the Hopper scenarios, they can be modeled using Constrained Markov Decision Processes (CMDP). We assume that the transition functions in these CMDPs are deterministic and in the form of the control-affine dynamics in equation (4). Each episode results in a trajectory  $\sigma := \{x_1, u_1, r_1, c_1, \dots, x_T, u_T, r_T, c_T\}$ , where the subscript represents the index of the sampling time.

The normalized reward and cost returns of a trajectory  $\sigma$  generated using policy  $\pi$  are defined as follows:  $R_{\text{normalized}}^{\pi} := \frac{R_{\sigma}^{\pi} - R_{\min}}{R_{\max} - R_{\min}} \times 100$  and  $C_{\text{normalized}}^{\pi} := \frac{C_{\sigma}^{\pi} + \varepsilon}{\kappa + \varepsilon}$ , where  $R_{\sigma}^{\pi} = \sum_t r_t$  denotes the policy’s reward return when generating trajectory  $\sigma$ ,  $C_{\sigma}^{\pi} = \sum_t c_t$  denotes its cost return, and  $R_{\min}$  and  $R_{\max}$  denote the maximum and minimum empirical reward returns over all trajectories in the dataset as defined and computed in DSRL [51]. In all benchmark tasks, the cost  $c_t$  is zero if the state  $x_t$  is safe and one if it is unsafe.

The parameter  $\kappa$  represents the cost threshold used to train the safe offline RL model, while  $\varepsilon$  is a small positive constant introduced to keep  $C_{\text{normalized}}^{\pi}$  well-defined when  $\kappa = 0$ . We set  $\kappa = 20$  and  $\varepsilon = 0$  in all of our experiments. In the case of FISOR,  $\kappa$  is not used for training the policy and

only used to compute the normalized cost. We say that a trajectory is safe when the normalized cost,  $C_{\text{normalized}}$ , satisfies the condition  $C_{\text{normalized}} < 1$ , implying that the cost return is less than the specified threshold when  $\varepsilon = 0$ .

### A.4.3 Hyperparameters

Table 6: Hyperparameters for the NCBF and dynamics model in the safety gymnasium environments

Parameter	Value
hidden dimension (CBF)	128
Number of hidden layers (CBF)	4
Learning rate (CBF)	$1 \times 10^{-5}$
$\varepsilon_{\text{safe}}$	0.08
$\varepsilon_{\text{unsafe}}$	0.15
$\varepsilon_{\text{ascent}}$	0.02
$w_{\text{ascent}}$	2
$w_{\text{safe}}$	1.0
$w_{\text{unsafe}}$	1.2
$w_{\text{lip}}$	1.0
Batch size	256
Hidden dimension (dynamics)	128
Number of hidden layers (dynamics)	4
Learning rate (dynamics)	$1 \times 10^{-4}$
Optimizer	Adam
Time step (dt)	0.1

For each of Swimmer and Hopper, we selected the best hyperparameters across three different models when reporting averaged results. In the Hopper environment, we choose the best  $w_c$  and  $\tau$  parameters for the CCBF models as  $(0.1, 1)$ ,  $(0.05, 1)$ , and  $(0.5, 0.7)$ , respectively. For the iDBF models in Hopper, we select  $p$  values of  $1 \times 10^{-10}$ ,  $1 \times 10^{-5}$ , and  $1 \times 10^{-8}$ , respectively. Similarly, in the Swimmer environment, we select the best  $w_c$  and  $\tau$  parameters for the three CCBF models as  $(1, 0.5)$ ,  $(1, 1)$ , and  $(1, 0.7)$ , respectively. For the iDBF models in Swimmer, we select  $p$  values of  $1 \times 10^{-6}$ ,  $1 \times 10^{-9}$ , and  $1 \times 10^{-12}$ , respectively. We also train the safety filters in Swimmer and Hopper for 15k and 50k steps respectively and choose the best checkpoints.

## B Ablation studies

In this section, we present two ablation experiments designed to analyze: (1) the performance of iDBF for different model sizes and density thresholds  $p$ , and (2) the effect of labeling all OOD states as unsafe when training a NCBF from offline data, as opposed to our method which simply lowers the safety score of these OOD states relative to the InD state from which they were sampled around, without the need to label them as unsafe.

### B.1 Influence of model size and density threshold $p$ on performance of iDBF

To remain faithful to the iDBF methodology and address whether model size may be causing the sub-par closed-loop performance observed in the safety gymnasium environments, we evaluated two iDBF architectures. The first is a medium-sized neural network with 4 hidden layers of 128 neurons each, and the second is a larger model with 5 hidden layers of 400 neurons each. Tables 7 and 8 report the corresponding results.

For the Hopper environment, our results show that increasing the model size reduces the normalized cost across all tested density thresholds  $p$ , but it also reduces the reward. This indicates that a larger model improves iDBF’s ability to classify safe states from the training data and differentiate them from unsafe states generated via the BC-safe policy. In contrast, the Swimmer environment exhibits

mixed effects: a larger model reduces both cost and reward when  $p = 10^{-8}$ , but leads to an increased cost and improved reward for  $p = 0.1$  and  $p = 10^{-4}$ . This shows that while iDBF still degrades the performance relative to the nominal policy without safety filters across both model architectures, the degradation is less severe when using the larger model.

Table 7: Evaluation results of normalized reward ( $R\uparrow$ ) and normalized cost ( $C\downarrow$ ) across different policies and a **medium-sized** iDBF model trained with 4 hidden layers, 128 neurons each, and with varying density threshold  $p$ . The  $\uparrow$  symbol denotes that the higher reward, the better. The  $\downarrow$  symbol denotes that the lower cost, the better.

Policy	None		iDBF ( $p = 0.1$ )		iDBF ( $p = 10^{-4}$ )		iDBF ( $p = 10^{-8}$ )	
	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$
<i>Swimmer</i>								
BC	2.26 $\pm$ 0.64	0.44 $\pm$ 0.03	3.24 $\pm$ 1.41	0.31 $\pm$ 0.05	5.82 $\pm$ 1.13	0.41 $\pm$ 0.04	5.45 $\pm$ 1.54	0.45 $\pm$ 0.03
BC-Safe	0.12 $\pm$ 0.05	0.43 $\pm$ 0.03	0.19 $\pm$ 0.03	0.42 $\pm$ 0.01	0.22 $\pm$ 0.03	0.47 $\pm$ 0.02	0.27 $\pm$ 0.03	0.50 $\pm$ 0.01
COptiDICE	1.65 $\pm$ 0.33	0.30 $\pm$ 0.04	8.06 $\pm$ 0.49	0.44 $\pm$ 0.02	10.43 $\pm$ 0.55	0.53 $\pm$ 0.03	11.71 $\pm$ 0.87	0.58 $\pm$ 0.01
BEAR-L	0.61 $\pm$ 0.11	0.16 $\pm$ 0.03	0.57 $\pm$ 0.23	0.15 $\pm$ 0.03	0.62 $\pm$ 0.10	0.13 $\pm$ 0.02	0.40 $\pm$ 0.08	0.14 $\pm$ 0.02
BCQ-L	2.82 $\pm$ 1.15	0.25 $\pm$ 0.08	9.83 $\pm$ 2.06	0.49 $\pm$ 0.07	9.53 $\pm$ 1.52	0.39 $\pm$ 0.02	11.07 $\pm$ 0.90	0.46 $\pm$ 0.02
<b>Average</b>	1.49	0.32	4.38	0.36	5.32	0.39	5.78	0.43
<i>Hopper</i>								
BC	0.19 $\pm$ 0.27	0.04 $\pm$ 0.02	3.87 $\pm$ 0.18	0.33 $\pm$ 0.06	7.49 $\pm$ 1.26	0.63 $\pm$ 0.13	2.28 $\pm$ 0.88	0.25 $\pm$ 0.06
BC-Safe	0.03 $\pm$ 0.02	0.57 $\pm$ 0.01	0.17 $\pm$ 0.07	0.54 $\pm$ 0.04	0.35 $\pm$ 0.09	0.58 $\pm$ 0.02	0.25 $\pm$ 0.06	0.60 $\pm$ 0.01
COptiDICE	0.01 $\pm$ 0.01	0.18 $\pm$ 0.01	1.15 $\pm$ 0.23	0.23 $\pm$ 0.01	0.17 $\pm$ 0.10	0.19 $\pm$ 0.01	0.40 $\pm$ 0.07	0.18 $\pm$ 0.01
BEAR-L	0.37 $\pm$ 0.01	0.16 $\pm$ 0.01	0.35 $\pm$ 0.01	0.16 $\pm$ 0.00	0.35 $\pm$ 0.01	0.16 $\pm$ 0.00	0.35 $\pm$ 0.01	0.16 $\pm$ 0.00
BCQ-L	3.09 $\pm$ 0.34	0.50 $\pm$ 0.04	4.46 $\pm$ 0.18	0.55 $\pm$ 0.05	4.16 $\pm$ 0.29	0.55 $\pm$ 0.01	3.20 $\pm$ 0.47	0.55 $\pm$ 0.01
<b>Average</b>	0.74	0.29	2.00	0.36	2.50	0.42	1.30	0.35

Table 8: Evaluation results of normalized reward ( $R\uparrow$ ) and normalized cost ( $C\downarrow$ ) across different policies and a **large-sized** iDBF model trained with 5 hidden layers, 400 neurons each, and with varying density threshold  $p$ . The  $\uparrow$  symbol denotes that the higher reward, the better. The  $\downarrow$  symbol denotes that the lower cost, the better.

Policy	None		iDBF ( $p = 0.1$ )		iDBF ( $p = 10^{-4}$ )		iDBF ( $p = 10^{-8}$ )	
	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$
<i>Swimmer</i>								
BC	2.26 $\pm$ 0.64	0.44 $\pm$ 0.03	6.79 $\pm$ 1.48	0.42 $\pm$ 0.03	8.41 $\pm$ 0.70	0.44 $\pm$ 0.05	4.00 $\pm$ 1.22	0.54 $\pm$ 0.04
BC-Safe	0.12 $\pm$ 0.05	0.43 $\pm$ 0.03	0.19 $\pm$ 0.01	0.43 $\pm$ 0.02	0.32 $\pm$ 0.03	0.50 $\pm$ 0.00	0.47 $\pm$ 0.09	0.49 $\pm$ 0.00
COptiDICE	1.65 $\pm$ 0.33	0.30 $\pm$ 0.04	10.09 $\pm$ 0.82	0.52 $\pm$ 0.02	14.13 $\pm$ 0.23	0.44 $\pm$ 0.01	5.36 $\pm$ 0.28	0.54 $\pm$ 0.00
BEAR-L	0.61 $\pm$ 0.11	0.16 $\pm$ 0.03	0.55 $\pm$ 0.15	0.11 $\pm$ 0.03	0.48 $\pm$ 0.05	0.14 $\pm$ 0.01	0.53 $\pm$ 0.05	0.13 $\pm$ 0.02
BCQ-L	2.82 $\pm$ 1.15	0.25 $\pm$ 0.08	9.46 $\pm$ 1.28	0.41 $\pm$ 0.07	12.47 $\pm$ 1.79	0.49 $\pm$ 0.06	8.90 $\pm$ 0.72	0.56 $\pm$ 0.03
<b>Average</b>	1.49	0.32	5.42	0.38	7.16	0.40	3.85	0.45
<i>Hopper</i>								
BC	0.19 $\pm$ 0.27	0.04 $\pm$ 0.02	0.91 $\pm$ 0.48	0.08 $\pm$ 0.03	2.56 $\pm$ 0.53	0.15 $\pm$ 0.05	3.34 $\pm$ 0.53	0.21 $\pm$ 0.04
BC-Safe	0.03 $\pm$ 0.02	0.57 $\pm$ 0.01	0.33 $\pm$ 0.07	0.59 $\pm$ 0.02	0.16 $\pm$ 0.08	0.51 $\pm$ 0.02	0.08 $\pm$ 0.05	0.45 $\pm$ 0.02
COptiDICE	0.01 $\pm$ 0.01	0.18 $\pm$ 0.01	0.21 $\pm$ 0.04	0.19 $\pm$ 0.01	0.06 $\pm$ 0.06	0.19 $\pm$ 0.00	0.01 $\pm$ 0.01	0.18 $\pm$ 0.00
BEAR-L	0.37 $\pm$ 0.01	0.16 $\pm$ 0.01	0.08 $\pm$ 0.05	0.20 $\pm$ 0.01	0.11 $\pm$ 0.00	0.16 $\pm$ 0.00	0.12 $\pm$ 0.01	0.16 $\pm$ 0.00
BCQ-L	3.09 $\pm$ 0.34	0.50 $\pm$ 0.04	2.30 $\pm$ 0.91	0.13 $\pm$ 0.05	2.67 $\pm$ 0.24	0.22 $\pm$ 0.02	2.55 $\pm$ 0.31	0.19 $\pm$ 0.05
<b>Average</b>	0.74	0.29	0.77	0.24	1.11	0.25	1.22	0.24

## B.2 Learning a NCBF from data labeled using an extremely conservative strategy

This ablation examines whether explicitly labeling OOD states as unsafe (as in iDBF) yields better results than softly lowering their safety scores relative to the InD safe state from which they were sampled around (as in CCBF). We used an extreme labeling approach: from each safe state in the dataset, we sampled random actions, took a one-step forward simulation, and marked all the resulting states as unsafe. This procedure mimics the effect of choosing a very high density threshold  $p$  within the iDBF framework. We call this approach **Extreme iDBF**. Results are shown in Table 9.

We tested this method with both the medium- and large-sized iDBF models to explore whether additional model capacity aids in learning better safety boundaries, given the inherent difficulty involved in this method where nearby states receive conflicting labels (safe and unsafe).

The empirical findings indicate that, for both model sizes, on average across all nominal controllers, this extreme labeling method results in poorer performance compared to the nominal policy without any safety filter. Notably, across both Swimmer and Hopper environments, pairing BEAR-L with this extreme version of iDBF achieves lower cost (with similar reward levels) than using BEAR-L without the safety filter. However, combining COptiDICE or BCQ-L with iDBF severely degrades performance, as evidenced by a significant increase in cost. We note that these findings align with those in [60] which suggest that assigning high-cost values to both OOD and unsafe actions can result in poor generalizability, which might explain the suboptimal performance observed during policy roll-outs when training a safety filter with this labeling strategy.

Table 9: Evaluation of normalized reward ( $R\uparrow$ ) and normalized cost ( $C\downarrow$ ) across different policies using iDBF with both **medium-** and **large-**sized models, each employing an **extreme labeling strategy (Extreme iDBF): every state reached by a one-step forward simulation with a randomly sampled action is labeled as unsafe irrespective of the density threshold  $p$** . The  $\uparrow$  symbol denotes that the higher reward, the better. The  $\downarrow$  symbol denotes that the lower cost, the better.

Policy	None		Extreme iDBF (medium-sized)		Extreme iDBF (large-sized)	
	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$
<i>Swimmer</i>						
BC	2.26 $\pm$ 0.64	0.44 $\pm$ 0.03	9.18 $\pm$ 1.05	0.55 $\pm$ 0.01	5.55 $\pm$ 1.02	0.46 $\pm$ 0.05
BC-Safe	0.12 $\pm$ 0.05	0.43 $\pm$ 0.03	0.54 $\pm$ 0.12	0.49 $\pm$ 0.01	0.19 $\pm$ 0.03	0.47 $\pm$ 0.02
COptiDICE	1.65 $\pm$ 0.33	0.30 $\pm$ 0.04	13.63 $\pm$ 0.65	0.59 $\pm$ 0.01	15.68 $\pm$ 0.31	0.48 $\pm$ 0.01
BEAR-L	0.61 $\pm$ 0.11	0.16 $\pm$ 0.03	0.41 $\pm$ 0.08	0.15 $\pm$ 0.02	0.54 $\pm$ 0.20	0.15 $\pm$ 0.02
BCQ-L	2.82 $\pm$ 1.15	0.25 $\pm$ 0.08	10.84 $\pm$ 0.72	0.45 $\pm$ 0.02	11.17 $\pm$ 0.45	0.44 $\pm$ 0.02
<b>Average</b>	1.49	0.32	6.92	0.45	6.63	0.40
<i>Hopper</i>						
BC	0.19 $\pm$ 0.27	0.04 $\pm$ 0.02	2.01 $\pm$ 0.28	0.21 $\pm$ 0.07	4.75 $\pm$ 0.47	0.17 $\pm$ 0.02
BC-Safe	0.03 $\pm$ 0.02	0.57 $\pm$ 0.01	0.43 $\pm$ 0.11	0.35 $\pm$ 0.03	0.08 $\pm$ 0.06	0.34 $\pm$ 0.04
COptiDICE	0.01 $\pm$ 0.01	0.18 $\pm$ 0.01	0.28 $\pm$ 0.04	0.10 $\pm$ 0.00	1.27 $\pm$ 0.26	0.23 $\pm$ 0.01
BEAR-L	0.37 $\pm$ 0.01	0.16 $\pm$ 0.01	0.12 $\pm$ 0.01	0.15 $\pm$ 0.00	0.19 $\pm$ 0.01	0.16 $\pm$ 0.00
BCQ-L	3.09 $\pm$ 0.34	0.50 $\pm$ 0.04	2.25 $\pm$ 0.40	0.10 $\pm$ 0.03	3.23 $\pm$ 0.44	0.52 $\pm$ 0.04
<b>Average</b>	0.74	0.29	1.02	0.18	1.90	0.28

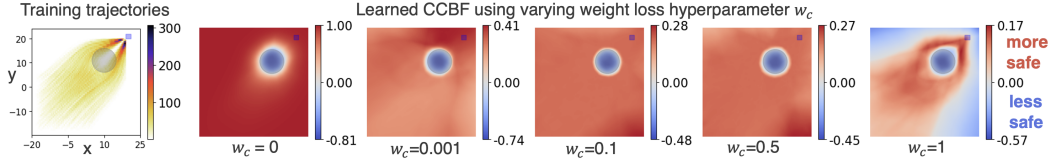
## C Additional discussion of results: safety barrier performance is dependent on the nominal controller and the environment

The closed-loop performance of a learned safety filter—whether based on NCBF, iDBF, or our proposed CCBF—is affected by the quality of the nominal controller. A learned barrier function that is not formally verified will often not satisfy the CBF condition in (2) for certain state-action pairs, and the nominal controller determines which actions at each state are examined by the QP solver. This seems to have a minimal effect in low-dimensional tasks such as the 2D navigation experiment, as all nominal controllers equipped with NCBF and CCBF achieve zero collisions and high success rate using any of the nominal controllers.

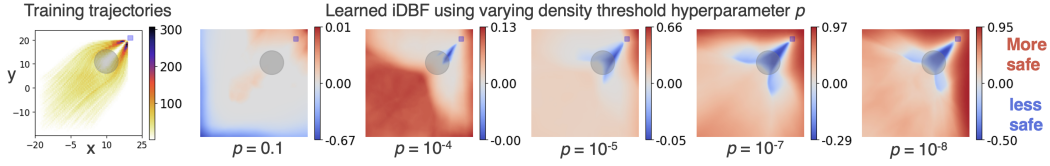
However, in the vision-based navigation experiment, the choice of nominal controller had a pronounced impact on performance. We can see in Table 1 how all the learned barrier functions perform poorly when paired with the goal reaching PD nominal controller. However, when paired with BC and BC-Safe nominal controllers, both iDBF and CCBF perform well, with CCBF achieving zero collisions and high success rate.

Interestingly, controllers in the safety gymnasium environment based on BC variants and offline reinforcement learning show varied behavior when augmented with a barrier. For instance, in the Hopper environment, when paired with BEAR-L, CCBF reduces the average cost from 0.37 to 0.20 and raises the average reward from 0.16 to 0.30. In the Swimmer environment with the BC-Safe controller paired with CCBF, the cost decreases from 0.12 to 0.03 while the reward increases from 0.43 to 0.45. For other controllers paired with CCBF, the decrease in cost returns is traded with a small decrease in reward returns.

## D Visual representation of the learned CCBF and iDBF using different hyperparameters in the 2D navigation environment



(a) Comparison of different CCBFs learned with different values of  $w_c$  shows that incorporating  $\mathcal{L}_c$  makes the CCBF more conservative than NCBF, while still preserving its ability to distinguish between safe and unsafe states. When  $w_c = 1$ , the CCBF effectively differentiates between InD and OOD states.



(b) Comparison of different iDBFs learned with different density thresholds  $p$  to select out-of-distribution actions highlights that iDBF is sensitive to the choice of  $p$ , and struggles to strike a proper balance between accurately separating safe and unsafe sets while respecting the underlying training distribution.

Figure 4: Visual comparison of learned CCBF and iDBF models with different hyperparameters.

## E Closed loop simulation results of CCBFs learned with and without detaching $B_\phi(x')$ in the computation of $L_c$ during training

Table 10: Evaluation results for different pairs of nominal controllers and CCBFs trained with and without detaching  $B_\phi(x')$  when computing  $L_c$  during training. Results of each pair are averaged across 500 runs for the 2D navigation task and 50 runs for the vision-based navigation task.

Policy	None		NCBF (without detach)		CCBF (with detach)	
	Success %	Collision %	Success %	Collision %	Success %	Collision %
<i>2D Navigation</i>						
PD controller	–	–	91.6±3.3	0	91.0±2.2	0
BC	96.2±2.2	21.8±4.2	96.4±2.3	0	95.4±2.6	0
BC-Safe	93.6±3.2	6.4±2.3	93.2±3.5	0	92.8±3.6	0
<b>Average</b>	94.9	14.1	93.7	0	93.1	0
<i>Vision-based Navigation</i>						
PD controller	–	–	68.0±17.9	82.0±8.4	58.0±14.8	4.0±5.5
BC	98.0±4.5	36.0±16.7	58.0±4.5	34.0±16.7	86.0±11.4	0
BC-Safe	92.0±8.4	22.0±4.5	62.0±4.5	24.0±8.9	94.0±8.9	0
<b>Average</b>	95.0	29.0	62.7	46.7	79.3	1.3

Table 11: Evaluation results of normalized reward ( $R\uparrow$ ) and normalized cost ( $C\downarrow$ ) for different pairs of nominal controllers and CCBFs trained with and without detaching  $B_\phi(x')$  when computing  $L_c$  during training. The  $\uparrow$  symbol denotes that the higher reward, the better. The  $\downarrow$  symbol denotes that the lower cost, the better. **Bold**: Safe agents (Cost < 1), otherwise agents are unsafe.

Policy	None		CCBF (without detach)		CCBF (with detach)	
	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$	Cost $\downarrow$	Reward $\uparrow$
<i>Swimmer</i>						
BC	2.26±0.64	0.44±0.03	<b>0.61±0.17</b>	<b>0.31±0.02</b>	<b>0.94±0.18</b>	<b>0.39±0.03</b>
BC-Safe	<b>0.12±0.05</b>	<b>0.43±0.03</b>	<b>0.07±0.01</b>	<b>0.38±0.01</b>	<b>0.03±0.02</b>	<b>0.45±0.02</b>
COptiDICE	1.65±0.33	0.30±0.04	<b>0.05±0.04</b>	<b>0.39±0.04</b>	<b>0.19±0.08</b>	<b>0.28±0.06</b>
BEAR-L	<b>0.61±0.11</b>	<b>0.16±0.03</b>	<b>0.48±0.09</b>	<b>0.13±0.02</b>	<b>0.42±0.17</b>	<b>0.09±0.01</b>
BCQ-L	2.82±1.15	0.25±0.08	2.91±1.31	0.32±0.03	1.40±0.47	0.29±0.03
<b>Average</b>	1.49	0.32	<b>0.82</b>	<b>0.31</b>	<b>0.59</b>	<b>0.30</b>
<i>Hopper</i>						
BC	<b>0.19±0.27</b>	<b>0.04±0.02</b>	<b>0.09±0.05</b>	<b>0.04±0.01</b>	<b>0.09±0.06</b>	<b>0.04±0.01</b>
BC-Safe	<b>0.03±0.02</b>	<b>0.57±0.01</b>	<b>0.16±0.11</b>	<b>0.58±0.02</b>	<b>0.05±0.03</b>	<b>0.56±0.05</b>
COptiDICE	<b>0.01±0.01</b>	<b>0.18±0.01</b>	<b>0.01±0.01</b>	<b>0.17±0.01</b>	<b>0.03±0.03</b>	<b>0.17±0.01</b>
BEAR-L	<b>0.37±0.01</b>	<b>0.16±0.01</b>	<b>0.18±0.01</b>	<b>0.15±0.01</b>	<b>0.20±0.06</b>	<b>0.30±0.06</b>
BCQ-L	3.09±0.34	0.50±0.04	1.43±0.15	0.31±0.04	1.16±0.53	0.37±0.05
<b>Average</b>	<b>0.74</b>	<b>0.29</b>	<b>0.37</b>	<b>0.25</b>	<b>0.31</b>	<b>0.28</b>

It can be seen that detaching  $B_\phi(x')$  in the computation of  $L_c$  during training of CCBFs leads to better performance compared to when not detaching it. In the 2D navigation task, both CCBFs (with and without detach) perform similarly, whereas CCBF (with detach) performs better in the vision-based navigation task. In both Swimmer and Hopper environments, CCBF (without detach) underperforms CCBF (with detach), but is still able to significantly improve the closed loop performance of the nominal controllers and outperforms NCBF and iDBF.