

Learning Conservative Neural Control Barrier Functions from Demonstrations

Ihab Tabbara¹ and Hussein Sibai²

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 for such filters, however, suffer from the curse-of-dimensionality. Deep learning approaches have been proposed in recent years to address this challenge. In this paper, we add to this set of approaches an algorithm for training neural control barrier functions from offline datasets. Such functions can be used to design constraints for quadratic programs that are then used as safety filters. Our algorithm trains these functions so that the system is not only prevented from reaching unsafe states but is also disincentivized from reaching out-of-distribution ones, at which they would be less reliable. 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 while minimally affecting task performance. Code is available at <https://github.com/tabz23/CCBF>.

I. INTRODUCTION

Ensuring that control systems satisfy safety specifications is critical in various applications, 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 forward invariance of sets of states disjoint from a user-defined *failure* set of states, such as collisions, while minimally altering (potentially unsafe) task-achieving nominal control [4], [5]. However, synthesizing CBFs using correct-by-construction techniques, such as sum-of-squares programming [6], [7], does not scale beyond few dimensions.

To address this challenge, researchers have proposed data-driven methods for learning safety filters [8], which have been applied in settings with uncertain dynamics [9], [10] and high-dimensional observations, such as images [11], [12], [13], [14] and point clouds [15], [16]. 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, i.e., belong to a forward invariant set disjoint from the failure set, or unsafe, i.e., belong to the complement of that forward invariant set, and then the filter gets retrained accordingly [17]. 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. 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 [18]. Several algorithms have been proposed to learn optimal policies from offline datasets of trajectories in that literature [19], [20], [21], [22]. 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 safe ones. For that purpose, we introduce Conservative Control Barrier Functions (CCBFs), a method for training neural CBFs that is inspired by conservative Q-learning (CQL) [19], an offline RL algorithm. The core insight is that the learned neural CBF should avoid over-estimating the safety of states and actions unseen in the dataset. Our method exhibits this property as seen in Figure 1. This objective is similar to the CQL objective, which minimizes the Q -values for randomly sampled actions at the dataset states in order to learn a Q -function that lower bounds the true one. That addresses the common problem of overestimating the Q -values for unseen state-action pairs in offline RL. By analogously minimizing the control barrier function's values of states reached by random actions starting from the dataset states, CCBFs avoid overestimating the safety of unseen states.

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.

II. RELATED WORK

a) Learning safety filters from expert demonstrations: Several algorithms have been proposed recently to train neural CBFs from offline datasets [10], [23], [24]. They differ in their objectives, their dataset generation process, their assumptions, and their guarantees, if any. [25] and [26] proposed learning safety filters to prevent control systems from reaching out-of-distribution (OOD) states. [25] proposed an algorithm for training neural CBFs, termed in-distribution

¹Ihab Tabbara and ²Hussein Sibai are in the Department of Computer Science at Washington University in St. Louis, MO 63130, USA.

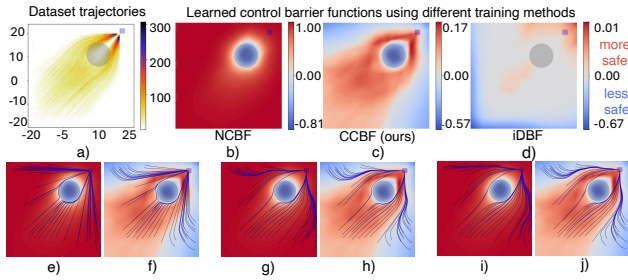


Fig. 1: Learned barrier functions and trajectory rollouts.

(a) Offline dataset trajectories with an obstacle (gray) and a goal (purple) (b–d) NCBF, CCBF (ours), and iDBF, respectively. (e–j) Rollouts (blue) using three CBF-QP policies, with nominal controllers ordered left to right as PD, BC, and BC-Safe (defined in Section IV-B). For each pair of figures, the left panel shows the results of using NCBF as the CBF in CBF-QP policy and the right panel shows the results of using CCBF instead. CCBF consistently balances safety with goal-reaching by accounting for both the obstacle and the training distribution, whereas NCBF emphasizes only obstacle avoidance and iDBF is very conservative.

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. Then, for each state in the training dataset, the algorithm randomly samples a set of actions, and if the BC policy assigns a sampled action a probability density below a user-defined threshold, 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., [9]) to train the iDBF using the augmented dataset, treating the InD states as safe and the OOD states as unsafe. [26] proposed Lyapunov Density Models, combining control Lyapunov functions [27] and density models [28], which are also learned from data to keep a control system from reaching OOD states. More recently, [29] 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 [30] 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 [31], [10].

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 unsafe states and disincentivize reaching OOD ones while minimally affecting task progress.

b) Safe offline RL: Safe offline RL is a branch of offline RL that aims to train policies that satisfy safety constraints while maximizing reward using offline datasets. 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 [32]. 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 failure states that we consider. Several approaches incorporated safety filters into the safe offline RL pipeline [33], [34]. A close work to ours is the recently proposed method FISOR [33], 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 (BRS) of a known constraint set (aka failure set) from the offline dataset. It uses Implicit Q-learning [35] 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 (belong to a controlled forward invariant set disjoint from the failure set) or unsafe, train a neural CBF with an additional loss term inspired from CQL [19] 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 safe offline RL algorithms.

III. PRELIMINARIES

In this paper, we consider nonlinear control-affine systems.

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.

A set of states $S \subseteq \mathcal{X}$ is called *controlled forward invariant* with respect to system (1) if there exists a control policy that makes it *forward invariant*, i.e., all trajectories starting in S remain inside it.

We assume that there is a user-defined set of failure states, such as collisions or violations of speed limits of autonomous vehicles, that a safe controller should prevent the system from reaching. Given a controlled forward invariant set S that is disjoint from the failure set, we say that the states in S are *safe* (and S is the safe set) and the states in $\bar{S} := \mathcal{X} \setminus S$ are *unsafe* (and \bar{S} is the unsafe set). Note that \bar{S} contains the failure set and might also contain additional states, particularly when there are input constraints or the system has a high relative degree [17], [36]. Thus, safety is defined as the invariance to a set disjoint from the failure set. The largest safe set S can be obtained using Hamilton-Jacobi (HJ) reachability analysis [36]. Unfortunately, existing algorithms for HJ reachability do not generally scale beyond a few dimensions [37].

A. Control barrier functions

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: $\forall x \in \mathcal{D} \subseteq \mathcal{X}$,

$$\exists u \in \mathcal{U} \text{ such that } \dot{B}(x, u) + \alpha(B(x)) \geq 0, \quad (2)$$

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

A CBF B for system (1) specifies the controls that guarantee the forward invariance of $B_{\geq 0}$, as stated in the following theorem. If the set of unsafe states is disjoint from $B_{\geq 0}$ and the system starts from $B_{\geq 0}$, 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 $\forall x \in \mathcal{D}, \pi(x) \in K_{\text{cbf}} := \{u \in \mathcal{U} \mid \nabla B(x)(f(x) + g(x)u) + \alpha(B(x)) \geq 0\}$ 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 denoted by a CBF-QP policy π_{safe} , as shown below:

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

Under some non-restrictive conditions, it can be proven that π_{safe} is locally Lipschitz continuous for $x \in \text{Int}(B_{\geq 0})$, the interior of the set $B_{\geq 0}$ [5].

B. Offline reinforcement learning

The goal of offline RL is to design optimal control policies from offline datasets without further interaction with the environment. Conservative Q learning (CQL) [19] is a prominent offline RL approach with several variants. The first variant, which is the one we are interested in here and simply call CQL for the rest of the paper, learns a Q function from the offline dataset corresponding to some policy that lower bounds its actual Q function pointwise for all state-action pairs, guaranteeing not over-estimating the values of state-action pairs that are not part of the dataset. Similarly, we want to learn barrier functions which do not over-estimate the safety of states not part of the dataset.

CQL modifies the Q -function learning iterative update for a policy π by adding a term to minimize the Q -values of a set of state-action pairs, where the states are those in the dataset and the actions are sampled from some distribution $\mu(a|s)$, to the standard Bellman error term, as follows:

$$\begin{aligned} \hat{Q}^{k+1} &\leftarrow \arg \min_Q \alpha \mathbb{E}_{s \sim \mathcal{D}, a \sim \mu(a|s)} [Q(s, a)] \\ &+ \frac{1}{2} \mathbb{E}_{s, a \sim \mathcal{D}} \left[Q(s, a) - \hat{B}^\pi \hat{Q}^k(s, a) \right]^2, \end{aligned} \quad (4)$$

where α is a hyperparameter, \mathcal{D} is the offline dataset, \hat{Q}^{k+1} denotes the learned Q -function at iteration $k + 1$, \hat{Q}^k is the learned Q -function from the previous iteration, and \hat{B}^π is the empirical Bellman operator for policy π .

Theorem 3.1 in [19] establishes that the fixed point $\hat{Q}^\pi = \lim_{k \rightarrow \infty} \hat{Q}_k$ obtained by the update in equation (4), when

the support of μ is a subset of the support of π_β (where π_β denotes the behavior policy that generated the offline dataset \mathcal{D}), is a pointwise lower bound of the true Q -function of policy π , i.e., $\hat{Q}^\pi(s, a) \leq Q^\pi(s, a), \forall (s, a)$, when α is sufficiently large. Thus, the learned Q -function \hat{Q}^π conservatively underestimates the true Q -function Q^π , preventing over-optimistic estimates for OOD actions. Previous research has already built a connection between control barrier functions and value functions [38], [17], so we adopt a similar analogy here to design a conservative loss for learning control barrier functions which do not over-estimate the safety of states unseen in the dataset.

C. Problem statement

We consider the setting where an offline trajectory dataset, that is generated using a potentially unknown and unsafe reference controller, is available, along with a user-specified failure set of states. We assume that the user has domain knowledge that allows them to label some, if not all, of the states in the dataset whether they belong to a (not exactly known) controlled forward invariant set that is disjoint from the user-specified failure set, i.e., safe, or not¹. Our objective is to train a neural CBF using the offline dataset that can be used to define the constraints in a quadratic program that acts as a safety filter that prevents the system from reaching the failure states. Consequently, it must also not over-estimate the safety of states unseen in the dataset which can result in trajectories reaching the failure set. Finally, the safety filter should not be conservative beyond what is necessary to maintain safety, i.e., as minimally altering the nominal control as possible.

IV. 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 their latent spaces to be the state spaces based on which we train the dynamics models and the neural CBFs. We discuss how we train each of these components next.

A. Learning latent dynamics

We train a deterministic 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 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

¹One can only label the states which belong to the provided failure set as unsafe and leave the rest of the states in the dataset unlabeled. However, that may lead to a neural CBF with a small or empty super-level set, which represents the safe set. That limits the states at which the system can start and the resulting safety filter be effective. Alternatively, we can adapt our method to training policy neural CBFs [17], which does not require the states to be labeled as safe or unsafe to train a neural CBF with a large super-level set, but we leave that for future work.

reconstructs them using a decoder \mathcal{G} . We train a control-affine dynamics model of the form:

$$x_{t+1} = x_t + (f_\theta(x_t) + g_\theta(x_t)u_t)\Delta t, \quad (5)$$

where f_θ and g_θ are multilayer perceptrons parametrized by θ , 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 (5), and λ_1 , λ_2 , and λ_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.

Generally, we assume that the continuous time dynamics are either user-provided or that they can be approximated in the form of (5) accurately on the training distribution, which is the case in our experiments. Our assumption that the dynamics are control-affine is needed so that the online safety filtering problem can be formulated as a quadratic program.

B. Learning nominal controllers

We train neural reference (nominal) controllers on the same offline dataset we use for training the CCBF. We use BC and safe offline RL algorithms to train these controllers. We train BC-based controllers by solving the optimization problem $\arg \max_{\beta} \mathbb{E}_{(x,u) \sim \mathcal{D}} [\log \pi_{\beta}(u|x)]$. When we only use the safe trajectories in the dataset in BC, we call the resulting policy BC-Safe. In the navigation environment, we use a PD controller defined as $\mathbf{u}_{\text{PD}} = K_p(\mathbf{x}_g - \mathbf{x}) + K_d \frac{\mathbf{u}_{\text{nom}}}{\Delta t}$, where \mathbf{x}_g is the goal position, and K_p, K_d are the proportional and derivative gains. For safety gymnasium environments [39], we train BC and safe offline RL policies using DSRL [40], which provides training datasets and implementations of RL algorithms. While such policies will be safer than the ones that result from traditional (non-safe) offline RL algorithms, they optimize for soft constraints and as we show in our experiments, become safer when coupled with a safety filter.

C. Learning CCBFs from offline datasets

We assume that a dataset of trajectories of the form $\sigma := \{x_1, u_1, x_2, u_2, \dots, x_T, u_T\}$ is available. We define $\mathcal{X}_{\text{safe}}$ as the set of states labeled as safe and $\mathcal{X}_{\text{unsafe}}$ as the set of states labeled as unsafe in the dataset. We denote the set of consecutive triplets (x, u, x') in the dataset in which $x' \in \mathcal{X}_{\text{safe}}$ by $\mathcal{D}_{\text{safe}}$ and the set of triplets in which $x \in \mathcal{X}_{\text{safe}}$ and $x' \in \mathcal{X}_{\text{unsafe}}$ by $\mathcal{D}_{\text{unsafe}}$. Given this dataset, we train a neural CCBF $B_\phi : \mathcal{X} \rightarrow \mathbb{R}$ by minimizing the following loss function: $\mathcal{L}_{\text{CCBF}}(\phi) = \mathcal{L}_{\text{safe}}(\phi) + \mathcal{L}_{\text{unsafe}}(\phi) + \mathcal{L}_{\text{ascent}}(\phi) + \mathcal{L}_{\text{descent}}(\phi) + \mathcal{L}_{\text{lip}}(\phi) + \mathcal{L}_c(\phi)$, where $\mathcal{L}_{\text{safe}}(\phi) = \frac{w_{\text{safe}}}{|\mathcal{X}_{\text{safe}}|} \sum_{x \in \mathcal{X}_{\text{safe}}} \sigma(\varepsilon_{\text{safe}} - B_\phi(x))$, $\mathcal{L}_{\text{unsafe}}(\phi) = \frac{w_{\text{unsafe}}}{|\mathcal{X}_{\text{unsafe}}|} \sum_{x \in \mathcal{X}_{\text{unsafe}}} \sigma(\varepsilon_{\text{unsafe}} + B_\phi(x))$,

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

Here, σ is the ReLU activation function, and $\varepsilon_{\text{safe}}, \varepsilon_{\text{unsafe}}, \varepsilon_{\text{ascent}}, \varepsilon_{\text{descent}}, w_{\text{safe}}, w_{\text{unsafe}}, w_{\text{ascent}}, w_{\text{descent}}, w_{\text{lip}}, w_c$, and τ are non-negative real numbers chosen as hyperparameters. The set V_x is a uniformly sampled set of actions from \mathcal{U} to evaluate at state x as well as the action taken in the dataset at x . The state x'_v is the state reached by the learned dynamics following action v at state x . When we train without using \mathcal{L}_c , we call the result 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). Specifically, $\mathcal{L}_{\text{descent}}$ penalizes violations of (2) during training. While it's hard to determine violations of (2) inside the safe and unsafe sets, we can infer the violations whenever a trajectory crosses from a state labeled as safe to one that is labeled as unsafe. Moreover, the loss term \mathcal{L}_{lip} drives the CCBF to be smooth. Finally, \mathcal{L}_c drives the CCBF to be conservative.

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 next state appearing in the training trajectory. The temperature parameter τ controls the smoothness of the soft maximum. As τ decreases, the soft maximum approaches the actual maximum. When $\tau = 1$, \mathcal{L}_c is similar to a loss term appearing in the objective function of learning the Q -function for a learned policy π_k in a variant of CQL described in equation (4) of [19]. That variant corresponds to choosing the sampling distribution μ in equation (4) in Section III-B of our paper to be the one that maximizes the argument of the argmin in the presence of a regularizer $\mathcal{R}(\mu)$, i.e.,

$$\begin{aligned} \mu &\leftarrow \arg \max_{\mu} \alpha \mathbb{E}_{s \sim \mathcal{D}, a \sim \mu(a|s)} [Q(s, a)] \\ &+ \frac{1}{2} \mathbb{E}_{s, a \sim \mathcal{D}} \left[Q(s, a) - \hat{B}^{\pi_k} \hat{Q}^k(s, a) \right]^2 + \mathcal{R}(\mu), \quad (6) \end{aligned}$$

where π_k is the policy obtained by (6) at iteration $k - 1$, $\mathcal{R}(\mu) = -D_{\text{KL}}(\mu, \rho)$, D_{KL} is the KL-divergence, and ρ is the uniform distribution over the actions. Thus, μ would be the regularized maximizer of the iterate \hat{Q}^k . Substituting that μ in equation (4) in Section III-B results in:

$$\begin{aligned} \hat{Q}^{k+1} &\leftarrow \arg \min_Q \alpha \mathbb{E}_{s \sim \mathcal{D}} \left[\log \sum_a \exp(Q(s, a)) \right] \\ &+ \frac{1}{2} \mathbb{E}_{s, a \sim \mathcal{D}} \left[Q(s, a) - \hat{B}^{\pi_k} \hat{Q}^k(s, a) \right]^2, \quad (7) \end{aligned}$$

as shown in equation (4) in [19]. In our setting, $\mathcal{L}_{\text{CCBF}}$ is analogous to the objective function in (7) where the Q -function is replaced with the barrier function B_ϕ , the Bellman error term is replaced with the terms of $\mathcal{L}_{\text{CCBF}}$ besides \mathcal{L}_c which penalize violations of the CBF conditions similar to how the Bellman error penalizes violations of the Bellman equation, and the sampling distribution for the actions in V_x

in \mathcal{L}_c is chosen to be the policy that maximizes safety (the barrier values) with a regularizer in the form of the distance (KL-divergence) to the uniform distribution. Formalizing this connection is an interesting future direction.

V. CASE STUDIES

In this section, we describe our experimental setup and present our results and key observations.

A. 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 [39], where agents must move as quickly as possible while adhering to velocity constraints. In each experiment, we present the results of rolling out the policy under different conditions: with no safety filter and with a NCBF-, an iDBF-, and a CCBF-based filter. We compare these methods with FISOR [33], in the safety gymnasium tasks. When training the NCBFs, iDBFs, and CCBFs, we use the same model architecture, the same set of common hyperparameters, and the same dynamics model. The only hyperparameters which differ between the different models are those specific to the different methods. Moreover, we train the NCBFs and CCBFs on the same datasets consisting of both safe and unsafe trajectories, whereas we train the iDBFs only on the safe states from safe trajectories in the dataset (also excluding the safe states in the unsafe trajectories), as we do for training BC-Safe policies.

a) 2D navigation: The agent has 2D single-integrator dynamics, has actuation limits that constrain the maximum speed in each dimension to be v_{max} , and has to reach a goal position while avoiding a static circular obstacle. Note that the agent can stop at any state by choosing zero control. Thus, the complement of the set of states that define the obstacle is a controlled forward invariant set. Consequently, we label a state as safe if the Euclidean distance d to the center of the obstacle is greater than or equal to $r + \epsilon$, where r is the obstacle’s radius and ϵ is a small positive constant. States with $d \leq r$ are labeled as unsafe, and states in between are not used for training. The small ϵ margin improves training stability, as has been observed in other works, e.g., [41].

b) 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 2D position.

c) Safety gymnasium locomotion: We test on SafetyHopper (11D state, 3D control) and SafetySwimmer (8D state, 2D control) from safety gymnasium [39]. Agents earn rewards for forward motion but incur costs if velocity exceeds user-defined thresholds. We picked these two tasks as challenging ones evident by the results in Table II which show that FISOR and other safe offline RL algorithms fail to balance the safety and task performance in them. We labeled states as unsafe if they exceeded the maximum velocity constraint,

and labeled them as safe otherwise. Thus, we only labeled states in the failure set as unsafe and assumed that all other states in the dataset belong to a controlled forward invariant set. We also explored another strategy: in trajectories that eventually enter the failure set, we ignore the states outside the failure set and label only the states inside it (i.e., those with velocity exceeding the user-defined limit) as unsafe. States in trajectories that never enter the failure set over their entire duration are instead labeled as safe. However, this approach reduced the number of trajectories to sample safe states from 135 out of 1686 trajectories for Swimmer and 130 out of 2240 for Hopper and led to poor safety-preserving performance. Due to page limitations, we summarize the key results of that experiment and refer the readers to Section V-B for definitions of normalized cost (C) and reward (R): In Swimmer, averaged across all nominal controllers, using CCBF as a safety filter resulted in $C=1.46$ and $R=0.31$ (compared to $C=1.49$ and $R=0.32$ for nominal controllers without safety filters; Table II), outperforming NCBF ($C: 2.38, R: 0.26$). In the case of Hopper, both NCBF and CCBF performed poorly, degrading the performance of BC and BC-Safe, while maintaining the same performance for COptiDICE. BEAR-L and BCQ-L simulations terminate after only 5-30 steps because these algorithms, when paired with the CCBF and NCBF, produced policies that quickly drove the agent into terminal states such as falling, unstable torso angles, or extreme joint positions. Importantly, CCBF consistently outperformed NCBF under both labeling strategies, but both NCBF and CCBF achieved significantly better safety-preserving performance when trained using the full DSRL dataset compared to the trajectory-filtered approach for labeling safe states, emphasizing the importance of more data, even if it means potentially mislabeling few states.

B. Results

We discuss two main observations: (1) CCBF-based filters significantly improve safety while minimally affecting performance, (2) CCBFs need less hyperparameter tuning. Then, we discuss the results of different ablation studies that help us isolate the effect of our conservative loss term \mathcal{L}_c . We report the results for the 2D navigation and vision-based navigation scenarios in Table I 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 report our results for the safety gymnasium tasks in Table II using the metrics: **normalized reward** and **normalized cost**, following the evaluation criteria in the safe offline RL literature [40], [42]. For both the Swimmer and the Hopper scenarios, they can be modeled as CMDPs. We assume that the transition functions are deterministic and in the form of the control-affine dynamics in (5). Each episode results in a trajectory $\sigma := \{x_1, u_1, r_1, c_1, \dots, x_T, u_T, r_T, c_T\}$. The normalized reward and cost returns of a trajectory σ generated using policy π are defined as follows: $R_{\text{normalized}}^\pi := \frac{R_\sigma^\pi - R_{\min}}{R_{\max} - R_{\min}} \times 100$ and $C_{\text{normalized}}^\pi := \frac{C_\sigma^\pi}{R_\sigma^\pi}$, where $R_\sigma^\pi = \sum_t r_t$ denotes the reward return of σ , $C_\sigma^\pi = \sum_t c_t$ denotes its cost return, and R_{\min} and

R_{\max} denote the maximum and minimum reward returns of all the trajectories in the dataset as defined and computed in DSRL [40]. In all benchmark tasks, the cost c_t is zero if the state x_t is safe and one if it is unsafe. We performed several ablations, but due to space constraints, we summarize some of their key findings and report average metrics across all nominal controllers for each task instead of presenting detailed results for each filter-controller combination. Complete tables with all experimental results and all hyperparameters used will be available in an extended version of this paper.

TABLE I: Controller evaluation results (S=Success%, C=Collision%). Results averaged across 500 runs (2D) and 50 runs (vision-based). **Green:** Safest agent with the highest average success % across all controllers.

Policy	None		NCBF		CCBF		iDBF	
	S	C	S	C	S	C	S	C
<i>2D Navigation</i>								
PD controller	100.0	90.4	91.6	0	91.0	0	0	100.0
BC	96.2	21.8	95.2	0	95.4	0	9.6	3.4
BC-Safe	93.6	6.4	93.8	0	92.8	0	7.8	0
Average	94.9	14.1	93.5	0	93.1	0	5.8	34.5
<i>Vision-based Navigation</i>								
PD controller	100.0	96.0	34.0	20.0	58.0	4.0	94.0	80.0
BC	98.0	36.0	68.0	6.0	86.0	0	96.0	14.0
BC-Safe	92.0	22.0	78.0	16.0	94.0	0	98.0	10.0
Average	95.0	29.0	60.0	14.0	79.3	1.3	96.0	34.7

TABLE II: 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. **Bold:** Safe agents ($C < 1$). **Gray:** Unsafe agents ($C \geq 1$). **Green:** Safest agent with the highest average success % across all controllers.

Policy	None		NCBF		CCBF		iDBF	
	$C\downarrow$	$R\uparrow$	$C\downarrow$	$R\uparrow$	$C\downarrow$	$R\uparrow$	$C\downarrow$	$R\uparrow$
<i>Swimmer</i>								
BC	2.26±0.64	0.44±0.03	5.05±2.84	0.4±0.06	0.94±0.18	0.39±0.03	3.80±0.31	0.36±0.01
BC-Safe	0.12±0.05	0.43±0.03	0.12±0.06	0.46±0.02	0.03±0.02	0.45±0.02	0.19±0.06	0.50±0.02
COptiDICE	1.65±0.33	0.30±0.04	1.40±0.39	0.07±0.01	0.19±0.08	0.28±0.06	11.75±2.99	0.57±0.03
BEAR-L	0.61±0.11	0.16±0.03	0.46±0.05	0.18±0.05	0.42±0.17	0.09±0.01	0.43±0.07	0.14±0.01
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
Average	1.49	0.32	2.34	0.26	0.59	0.3	5.1	0.39
FISOR	0.01±0.01	-0.08±0.01	-	-	-	-	-	-
<i>Hopper</i>								
BC	0.19±0.27	0.04±0.02	0.23±0.18	0.04±0.01	0.09±0.06	0.04±0.01	4.71±2.26	0.33±0.24
BC-Safe	0.03±0.02	0.57±0.01	0.14±0.10	0.61±0.01	0.05±0.03	0.56±0.05	0.21±0.13	0.56±0.03
COptiDICE	0.01±0.01	0.18±0.01	0.01±0.01	0.17±0.01	0.03±0.03	0.17±0.01	1.81±0.73	0.24±0.03
BEAR-L	0.37±0.01	0.16±0.01	0.27±0.09	0.21±0.08	0.20±0.06	0.30±0.06	0.34±0.02	0.16±0.01
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
Average	0.74	0.29	0.39	0.27	0.31	0.28	2.18	0.35
FISOR	0.03±0.04	0.08±0.05	-	-	-	-	-	-

a) *CCBF-based filters improve safety while minimally affecting reward returns:* Observing the results in Table I 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. Trajectories taken when using CCBF and NCBF as a safety filter are presented in Figure 1.

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 low costs in the Swimmer and Hopper tasks (comparable to CCBF paired with BC-safe in the Swimmer task and with COptiDICE in the Hopper task). However, FISOR yields much lower rewards in both environments: -0.08 in Swimmer compared to 0.45 in the case of CCBF paired with BC-Safe, and 0.08 in Hopper compared to 0.17 when CCBF is paired with COptiDICE.

It is worth noting that the policies used during evaluation in Table II are distinct from those used to collect the DSRL dataset. This demonstrates that CCBFs, despite being trained on offline data, can effectively improve safety for new policies without noticeably affecting task performance.

iDBF performs well with the BC-Safe controller, improving safety in vision-based navigation and maintaining comparable levels in Swimmer and Hopper. This is expected since iDBF’s approach labels states as unsafe when they correspond to low-probability actions under BC-Safe. However, with other controllers such as COptiDICE, performance degrades, likely because iDBF steers them toward regions that BC-Safe deems safe but that are suboptimal if the dataset trajectories are not expert (reward-maximizing).

To further investigate iDBF’s performance, we conducted several ablation experiments by varying the model size and the density threshold p for selecting OOD actions. We tested medium-sized models (4 layers, 128 neurons) versus large models (5 layers, 400 neurons) across different density thresholds ($p \in \{1, 0.1, 10^{-4}, 10^{-8}\}$). For medium-sized models, average costs increased substantially across all thresholds compared to not using any safety filter: average Swimmer costs rose from 1.49 to 4.38-6.92, while Hopper costs

increased from 0.74 to 1.02-2.50. Large models generally showed improvement with average costs ranging from 3.85-6.63 for Swimmer and 0.77-1.90 for Hopper, demonstrating better performance than medium-sized models across most density thresholds. However, across all configurations and model sizes, iDBF’s performance consistently deteriorated and underperformed compared to nominal controllers, and to CCBF and NCBF methods that leverage actual unsafe trajectories in the offline dataset.

b) *CCBFs need less hyperparameter tuning*: iDBF’s performance was sensitive to the density threshold p used to classify sampled actions as OOD. We also had to tune the hyperparameters w_{safe} , w_{unsafe} , $\varepsilon_{\text{safe}}$, and $\varepsilon_{\text{unsafe}}$ to obtain an iDBF that does not assign a single value to all states. In the case of CCBF, we had to tune w_c to a value between 0.5 and 1 for the learned CCBF to disincentivize unseen actions, as shown in Figure 1. However, in all experiments, CCBFs distinguished safe and unsafe states when assigning w_c to any value between 0 and 1 without collapsing, unlike iDBF.

c) *Isolating the effect of our conservativeness-inducing loss term \mathcal{L}_c* : We conducted an experiment where we trained CCBFs with $w_c \in \{0, 0.5, 1.0\}$, for the vision-based navigation, Swimmer, and Hopper environments. Note that when $w_c = 0$, our approach is equivalent to training a NCBF. We split the dataset trajectories into training and testing ones and only train on the former. For each safe state in the test trajectories, we evaluated the CCBF at the next state along the trajectory as well as at a set of states obtained by taking random actions from that same state, following the procedure used to compute \mathcal{L}_c . First, we found that increasing w_c decreases the CCBF values of the states reached by taking both the actions in the test set and the actions randomly sampled, as expected. Second, increasing w_c enlarges the gap between the values of the CCBFs at the states in the test set compared to the ones reached by the random actions. This clearly shows that the learned CCBFs generalize well to distinguish actions taken by the policies generating the data and random ones and consider the states resulting from the latter as less safe. In contrast, we found that NCBFs consistently assign similar values for the randomly-reached states and the ones in the test set, also as expected. We present examples of our results in Figure 2.

Next, we studied whether other conservativeness-inducing loss terms would result in similar improvement in performance compared to the NCBF approach. We denote the first candidate loss term, which replaces \mathcal{L}_c , by $\mathcal{L}'_{\text{unsafe}}$ and its associate weight by w'_{unsafe} . The term $\mathcal{L}'_{\text{unsafe}}$ is equivalent to $\mathcal{L}_{\text{unsafe}}$ evaluated at the states sampled in the same manner when computing \mathcal{L}_c , i.e., treating the states reached by randomly-sampled actions similarly to unsafe ones. We call this new approach CCBF*. It exhibits similar properties to CCBF when comparing the values it assigns for the OOD and InD states in new trajectories and performs better than iDBF and NCBF, but underperforms compared to CCBF. We summarise the main results below due to lack of space. For Swimmer, the average results across controllers are: CCBF* with $w_c^* = 0.5$ ($C = 1.33$, $R = 0.34$) and $w_c^* = 1$ ($C = 2.44$,

$R = 0.32$). For Hopper, the averages are: CCBF* with $w_c^* = 0.5$ ($C = 0.37$, $R = 0.31$), and $w_c^* = 1$ ($C = 0.91$, $R = 0.26$). We note that the learned CCBF* resulting from $w_c^* = 1$ underperforms the one resulting from $w_c^* = 0.5$. This potentially occurs because $w_c^* = 1$ penalizes OOD states as much as actual unsafe states. Since randomly reached states are likely (but not necessarily) OOD, this aligns with findings in [43] stating that treating OOD and unsafe actions similarly degrades performance. As a remark, in both CCBF and CCBF*, increasing w_c and w_c^* above the value of one can result in over-conservative safety filters.

From these two experiments, we conclude that CCBF’s and CCBF*’s conservativeness-inducing loss terms enable learning safety filters that not only filter actions for safety, but also for staying in-distribution. Our experiments show that CCBF outperforms CCBF*, but further evaluation is needed to claim that one of these approaches is consistently better.

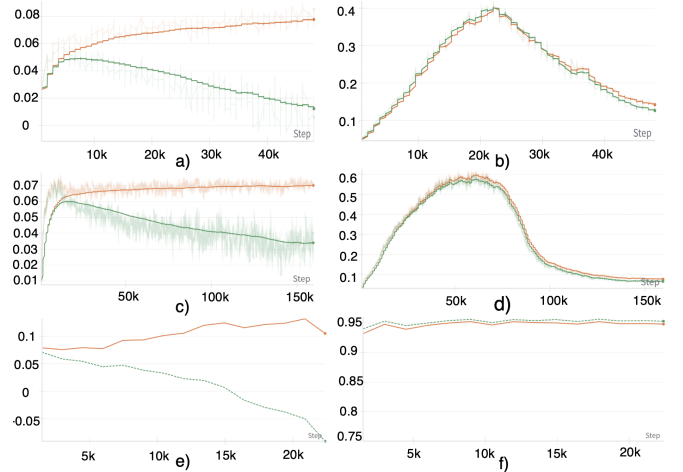


Fig. 2: Variation of the mean of the barrier values of states reached by taking either dataset actions (orange) or randomly sampled actions (green) starting from safe states in unseen test trajectories, plotted against the number of training steps. Results are shown for CCBF trained with $w_c = 0.5$: (a) Swimmer, (c) Hopper, and (e) vision-based navigation, where dataset actions are consistently evaluated as safer than random actions. For NCBF: (b) Swimmer, (d) Hopper, and (f) vision-based navigation, where this distinction is not observed and the two curves largely overlap.

VI. CONCLUSION

We introduced Conservative Control Barrier Functions (CCBFs), neural safety filters trained using offline data. The core idea is to add a loss term that prevents over-estimation of the safety of state-action pairs not seen in the dataset. When reference controllers and safety filters are trained offline, CCBFs, even when paired with reference controllers that were not used to generate the offline dataset, outperform baselines and result in closed-loop behaviors that better preserve safety while minimally impacting task performance. Our approach in its present form does not offer formal guarantees for safety maintenance or OOD state avoidance, and deriving ones would be exciting future work. Also, evaluating our method on maintaining the safety of real robots would also be needed for further validation.

REFERENCES

- [1] J. Betz, A. Heilmeier, A. Wischnewski, T. Stahl, and M. Lienkamp, "Autonomous driving—a crash explained in detail," *Applied Sciences*, vol. 9, no. 23, 2019. [Online]. Available: <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*, vol. 1, no. 2, pp. 65–76, 2019.
- [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*, vol. 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)*, 2019, pp. 3420–3431.
- [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*, vol. 62, no. 8, pp. 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*, vol. 10, no. 4, pp. 709–729, Apr. 2016. [Online]. Available: <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)*. IEEE, 2021, pp. 6105–6112.
- [8] 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*, vol. 39, no. 3, pp. 1749–1767, 2023.
- [9] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conference on Robot Learning*. PMLR, 2022, pp. 1724–1735.
- [10] 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.
- [11] 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)*. IEEE, 2023, pp. 782–788.
- [12] 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.
- [13] K. Nakamura, L. Peters, and A. Bajcsy, "Generalizing safety beyond collision-avoidance via latent-space reachability analysis," *arXiv preprint arXiv:2502.00935*, 2025.
- [14] I. Tabbara, Y. Yang, A. Hamzeh, M. Astafyev, and H. Sibai, "Designing latent safety filters using pre-trained vision models," *arXiv preprint arXiv:2509.14758*, 2025.
- [15] 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.
- [16] 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*, vol. 9, no. 2, pp. 1043–1050, 2023.
- [17] 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)*, 2024, pp. 11 532–11 539.
- [18] 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.
- [19] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.
- [20] Y. Wu, G. Tucker, and O. Nachum, "Behavior regularized offline reinforcement learning," *arXiv preprint arXiv:1911.11361*, 2019.
- [21] 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*, vol. 32, 2019.
- [22] 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.
- [23] 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.
- [24] 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)*. IEEE, 2024, pp. 11 532–11 539.
- [25] 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*. PMLR, 2023, pp. 286–299.
- [26] 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*. PMLR, 2022, pp. 10 708–10 733.
- [27] E. D. Sontag, "Control-lyapunov functions," in *Open problems in mathematical systems and control theory*. Springer, 1999, pp. 211–216.
- [28] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos, "Count-based exploration with neural density models," in *International conference on machine learning*. PMLR, 2017, pp. 2721–2730.
- [29] 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.
- [30] N. Charoenphakdee, Z. Cui, Y. Zhang, and M. Sugiyama, "Classification with rejection based on cost-sensitive classification," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1507–1517.
- [31] 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)*. IEEE, 2020, pp. 3717–3724.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: <https://openreview.net/forum?id=68n2s9ZJWF8>
- [36] 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)*. IEEE, 2017, pp. 2242–2253.
- [37] I. M. Mitchell, "The Flexible, Extensible and Efficient Toolbox of Level Set Methods," *Journal of Scientific Computing*, vol. 35, no. 2, pp. 300–329, Jun. 2008. [Online]. Available: <https://doi.org/10.1007/s10915-007-9174-4>
- [38] D. C. Tan, F. Acero, R. McCarthy, D. Kanoulas, and Z. Li, "Value functions are control barrier functions: Verification of safe policies using control theory," *arXiv preprint arXiv:2306.04026*, 2023.
- [39] 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*, vol. 36, pp. 18 964–18 993, 2023.
- [40] 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.
- [41] 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*, vol. 7, no. 3, pp. 7357–7364, 2022.
- [42] 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.
- [43] 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.