A ENVIRONMENTS

A.1 F1/10 RACE CAR

This simulator contains an agent that moves along a two-dimensional racetrack, which is modeled after well known F1 tracks downscaled to 1:10, as used in O'Kelly et al. (2020). The racetrack is assumed to be 2 m wide, and the observation space for the agent is two dimensional, reporting the distance to centerline, as well as the relative angle from it. At every step, the agent takes an action $a \in [-1, 1]$rad which indicates the steering angle. We use three tracks in our experiments: *Playground*, *Silverstone*, and *Austin*, which can be visualized through Figure 3(a)-(c).

To generate expert trajectory data from this environment, we create an expert planner using search-based model predictive control (MPC) which is able to generate collision-free paths between randomly sampled start and goal states within the track. To delibrately produce unsafe demonstrations, we randomly decrease the safety threshold in the MPC planner to generate trajectories that will crash. We generate 1k trajectories with 711 safe trajectories and 289 unsafe trajectories. We randomly sample 800 trajectories for training, and use the rest 200 trajectories for evaluation. The average collected trajectory length is 100.

A.2 MUSHR CAR SIMULATOR

MuSHR is a robot car equipped with a 2-D LiDAR sensor. The LiDAR sensor scans the environment around the car using 720 laser beams (with an angular resolution of $0.5\,\mathrm{deg}$) and returns an observation of shape $[720, 2]$, where each element is the x,y coordinate to the closest surface for that ray angle. Similar to before, the MuSHR car also takes a steering angle as the input action, which is of the range $a \in [-0.34, 0.34]$ rad in the expert demonstrations.

We create a simulator for this vehicle which takes a 2D occupancy map as an input, and instantiates the vehicle dynamics and the sensor model within it. We use a pre-mapped 2D office environment for the simulation which can be visualized in Figure 3(d). We build a probabilistic roadmap over the environment and sample start and goal states in the free space, from which we generate 10K trajectories in total, where each trajectory spans around 110 timesteps. Similar to above, we use a search-based MPC for computing the collision-free trajectories. We apply slight perturbations to the expert planner to also occasionally result in unsafe trajectories. We generate 7550 safe demonstrations and 2450 unsafe demonstrations. However, as shown in Supplementary C.7, ConBaT can be trained with just 0.01X of the unsafe demonstrations (21 trajectories) here to outperform almost all the baselines. The collected demonstration is shown in Figure 7

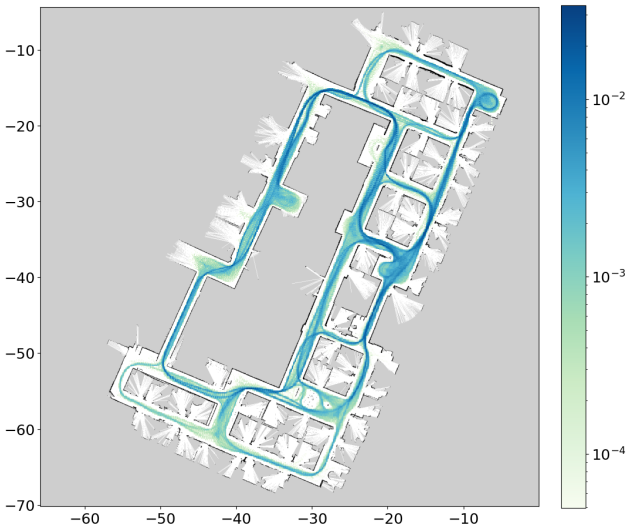


Figure 7: Visualization of the MuSHR environment occupancy map and collected data distribution

B IMPLEMENTATION

B.1 CONBAT IMPLEMENTATION

Transformer and Control Barrier Critic:
For both the F1/10 simulator and for MuSHR, we use a linear embedding to convert the observation and action into state and action tokens respectively. For F1/10, we train the policy and world model in phase 1 for 10 epochs, and then further train the control barrier critic for 10 epochs. For Mushr data, we train the policy for 50 epochs and further train the control barrier critic for 10 epochs. We use the Adam optimizer for training. Hyperparameters used for model training can be seen in Table 3.

Hyperparameter	Value
# of layers	2
# of attention heads	8
Embedding length	64
Sequence length	16
Batch size	32
Learning rate	1e-4
Optimizer	Adam
# of cbf layers	2
# of cbf units	128
γ	1.0
α	0.1
λ_c	1
λ_s	5
λ_f	1

Table 3: Hyperparameters

Online Optimization:
During deployment, the general aim is to be able to roll out a safe trajectory for as long as possible. At a high level, ConBaT is first given a short sequence of state-action pairs as a prompt. Based on this prompt and the current observation, the model predicts the next action, which is sent to the simulator to get the observation at the next time step. This roll out procedure of generating new action and new state is carried on iteratively until a crash happens or a maximum number of timesteps is reached. At timestep t of the rollout procedure, given the observed the history state and action sequence $(s_{t-T+1}, a_{t-T+1}, s_{t-T+2}, a_{t-T+2}, ..., s_{t-1}, a_{t-1}, s_t)$, we first compute the action proposal for time t. Then we combine the action proposal $\hat{a}_t$ with a learnable parameter Δa and forward them to the network. Initially, Δa is set to zero. We evaluate the violation loss for the CBF condition at time t as:

$$\mathcal{L}_v = \sigma_+(\eta - C_f(s_t^+, a_t^+)) \tag{8}$$

where η is the conservative threshold. η being positive means the CBC value should be greater than a positive value to satisfy the CBF condition, hence being more conservative. We only do online optimization if $\mathcal{L}_v$ is nonzero. We use the RMSProp optimizer and do $1 \sim 3$ backward steps to compute Δa. A detailed ablation study for several online optimization configurations can be seen in Appendix C

B.2 BASELINES

MPC: We implemented an optimization-based model predictive control baseline using the CasAdi solver. At every step, we crop a $5m \times 5m$ neighborhood centered at the MuSHR car from the map, and convert the occupied cells in the neighborhood to circular obstacles. To alleviate the planning burden, we only consider the obstacles that contain both occupied cells and free cells (which means the obstacle is at the boundary of the lidar scan). Then we perform $3 \sim 10$ steps of MPC planning to make sure the car is not colliding with any of the obstacles at any time in the MPC horizon. Our best approach involves a 10-step MPC horizon and choose 0.2m as the collision threshold (the car should be at least 0.2m far from the obstacle). After some grid-search, we find that this configuration gives us the best performance.

SAC: We train a Soft-Actor-Critic approach (Haarnoja et al., 2018) for 1000 epochs, where under each epoch, the model rolls out 1000 steps from the simulation and gets 1000 back-propagation updates. The policy net is a 3-layer fully-connected network with hidden units [128, 128, 128] for the layers. The gamma is 0.99, the learning rate is 1e-3, and the batch size is 256.

TRPO: We train a TRPO approach (Schulman et al., 2015) for 1000 epochs, where under each epoch the model rolls out 1000 steps from the simulation and gets 1000 back-prop updates. The policy net is a 3-layer fully-connected network with hidden units [128, 128, 128] for the layers. The discount factor gamma is 0.99, the learning rate is 3e-4, and the batch size is 1000.

PPO: We train a PPO approach (Schulman et al., 2017) for 1000 epochs, where under each epoch the model rolls out 1000 steps from the simulation and gets 1000 back-prop updates. The policy net is a 3-layer fully-connected network with hidden units [128, 128, 128] for the layers. The discount factor gamma is 0.99, the learning rate is 3e-4, and the batch size is 4000.

For all the RL algorithms, the agent receives a reward of 0.1 if it does not collide with anything, and -3 if it does.

GAIL: We train a GAIL approach (Ho & Ermon, 2016) for 1000 epochs, where we only use the safe set of the expert trajectories that were used in ConBaT and we train for 10000 iterations. The GAIL framework consists of a value network, a generator (policy net), and a discriminator. The value network is a 2-hidden-layer fully-connected network with 128 hidden units in each layer and the learning rate is 1e-3. For the generator, we use a 2-hidden-layer fully-connected network with 128 hidden units in each layer. The discount factor gamma for the generator is 0.995 and the learning rate is 3e-4. For the discriminator, we use a 2-hidden-layer fully-connected network with 128 hidden units in each layer, with the learning rate as 1e-4. The sample batch size is 4000.

BC: We train a 3-hidden-layer fully-connected network with 128 hidden units in each layer for 100 epochs, with batchsize 32 and a learning rate of 1e-4, using the same set of safe data used by GAIL.

Algorithm	Training time (h/m)	Runtime (s)
MPC	-	35580
BC	8h 6m	579.51
PPO	24h	560.87
TRPO	24h	558.62
SAC	24h	565.24
GAIL	24h	546.09
PACT	11h 2m	666.51
ConBaT	12h 5m	852.31

Table 4: Training and deployment time taken

C Additional Results

C.1 Computational Effort

In Table 4, we outline the computational requirements for the different classes of algorithms we implement in the MuSHR domain. We note MPC to be 1-2 orders of magnitude slower than the learning-based methods during runtime as it requires solving a complex optimization problem at every step, parameterized over the number of obstacles in the neighboring map. While the reinforcement learning baselines are relatively faster in deployment than PACT or ConBaT, their training time is much higher.

C.2 CBF Critic Ablations

In Table 5, we compare two different architecture designs for the future state C_f: CBC-EF (CBC-embedding/future) which takes state and action embeddings as input: $\hat{c}_{t+1} = C_f(s_t^+, a_t^+)$, and CBC-TF (CBC-token/future) which takes state embedding and action tokens as input: $\hat{c}_{t+1} = C_f(s_t^+, a_t')$. As shown in Table 5, the online optimization time for CBC-TF is $40\% \sim 60\%$ shorter

Track	Collision Rate (%)		ATL (# steps)		Runtime (sec)	
	E-CBC	T-CBC	E-CBC	T-CBC	E-CBC	T-CBC
Playground	0	1.5	1000	983.46	134.17	76.78
Silverstone	0	54.6	1000	632.28	147.61	81.94
Austin	61.7	96.8	678.15	279.13	180.88	76.62

Table 5: Using action embedding vs. action token for the Control Barrier Critic

than the runtime of CBC-EF, which is because the action token is not temporally-fused with features from other timesteps, whereas action embedding is, hence when performing backpropagation at action-embeddings (in CBC-EF's case) the computational burden increases. However, consistently among all the three tracks, CBC-EF achieves much better collision rate and ATL compared to CBC-TF, which we attribute to the expressiveness of the action embeddings after fusing with other state-action history. Thus, we primarily use the CBC-EF architecture in our paper.

C.3 Online Optimization Ablations

Table 6 contains the results from several ablation studies we perform on the F1/10 dataset to identify how the online optimization routine behaves under varying influence of its hyperparameters. We vary these three parameters: a) the number of gradient descent steps, b) the learning rate, and c) the CBF threshold defined as the optimal critic cost value. From Table 6(a), we note that increasing the optimization steps in the online optimization does not improve the performance. From Table 6(b), we see that a fairly small learning rate can already make the online optimization achieve collision-free performance on F1/10, whereas a larger learning rate leads to a more unstable optimization process hence deteriorating the result. From Table 6(c), we see that the threshold η to some degree reflects the conservativeness - a larger threshold will result in a more conservative result, which potentially can lead to better performance.

Steps	Collision	ATL
0	1	175.45
1	0.0078	991.24
2	0.0078	991.22
3	0.0078	991.21
4	0.0078	991.21

(a) SGD step.

lr	Collision	ATL
0.05	0	999
0.1	0.0078	991.23
0.2	0.0078	991.22
0.3	0.0156	983.4
0.5	0.0234	975.73

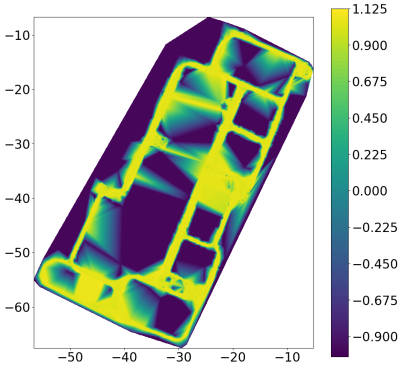
(b) Learning rate (lr)

Threshold	Collision	ATL
-0.1	0.875	343.6
-0.05	0.3672	714.08
0.0	0.0469	958.29
0.05	0.0156	983.47
0.2	0.0078	991.22

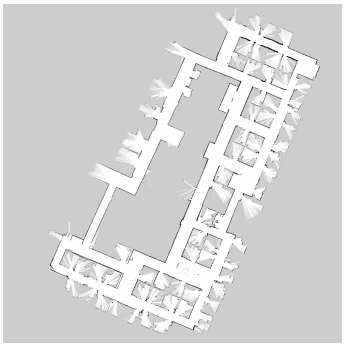
(c) Threshold

Table 6: Ablation studies for online optimization (iterations, learning rates and thresholds)

C.4 Learned Control Barrier Critic Visualization



(a) Visualization from Control Barrier Critics (CBC)

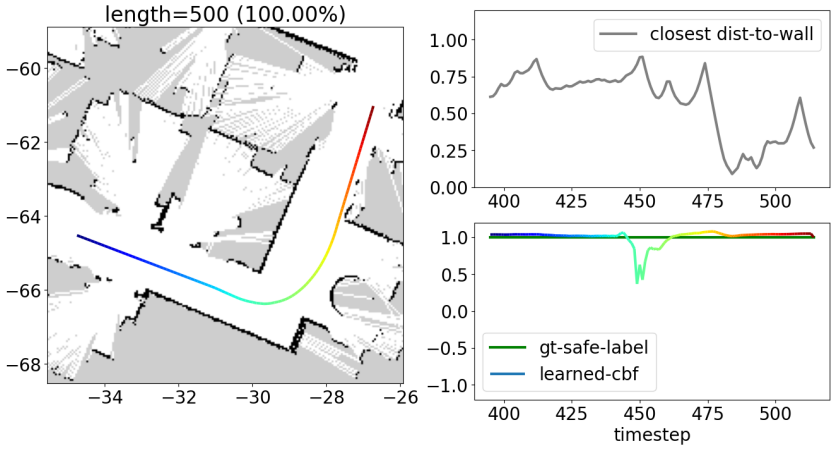


(b) Visualization of the original MuSHR environment

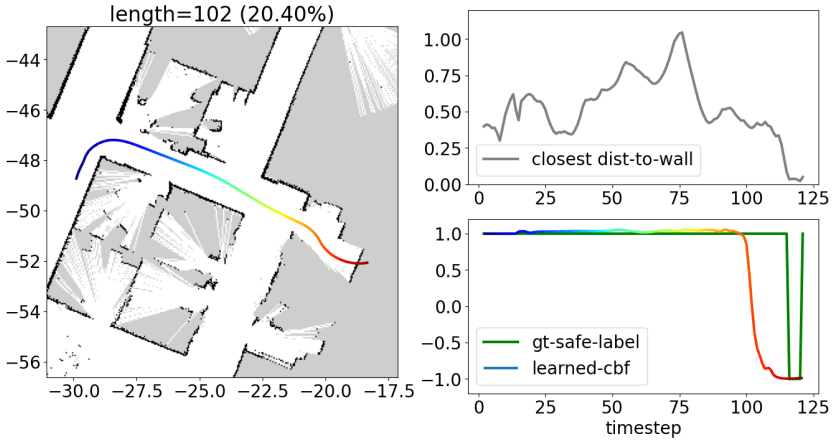
To inspect how the control barrier critic (CBC) is learned, we plot the CBC prediction along the expert trajectories and interpolate over unvisited regions on the map. As shown in 8a, the CBC is able to predict safe values in the center of the pathways, and predict negative values in regions that are close to the boundary of the wall, which is consistent with the expectation that the closer to the center of the hallway, the safer. However, we do notice that there exists unsafe area that the CBC mistakenly marks that area as "safe" (e.g., around the location (-30, -20)), which could be due to interpolation error or network inability to predict the safety score at that spot.

C.5 CBF VISUALIZATION

To illustrate how well the CBF is learned, we visualize the simulation as well as the observation/cbf/safety-label sequences along the trajectories. As shown in Fig. figure 9, in most of the cases, the CBF value is postive for safe states and negative for unsafe states, which shows that our learned CBF can correctly classify the safe/unsafe states. Besides, the decreasing trend (timestep 100-120 in Fig. figure 9b) of CBF can alert the potential collision in near future (however in this case, due to control limitation, the agent cannot escape from the collision).



(a) Safe case: The CBF value is always greater than 0 for safe states. The CBF value is not just remembering the closest distance-to-wall, but also quantifies the difficulty to escape from the potential future collision (timestep 450).



(b) Unsafe case: The CBF value will be negative when comes to collision (timestep 120-125), and the decreasing trend (timestep 100-120) of the CBF value can alert the potential collision in near future.

Figure 9: Visualization for the simulation and measurement/cbf/safety-label trajectories

C.6 How does the negative samples affect the testing performance

Although ConBaT requires negative demonstrations in training, it turns out only a few negative samples will suffice to guide the CBF-critic training and to improve the rollout performance. Here we train ConBaT under different number of negative samples (ranging from 21 to 2131) and evaluate their 5000-step rollout trajectories under 128 initial conditions. We measure the rollout performance by safety rate, i.e., the percentage of rollout trajectories that are safe. As shown in Fig. 9, even just using 21 negative samples in training, the success rate of ConBaT already outperforms almost all the baselines except SAC (which is 1% higher). By leveraging 106 negative examples, we already surpass all the baselines considered in this paper. This showcases our algorithm is data-efficient.

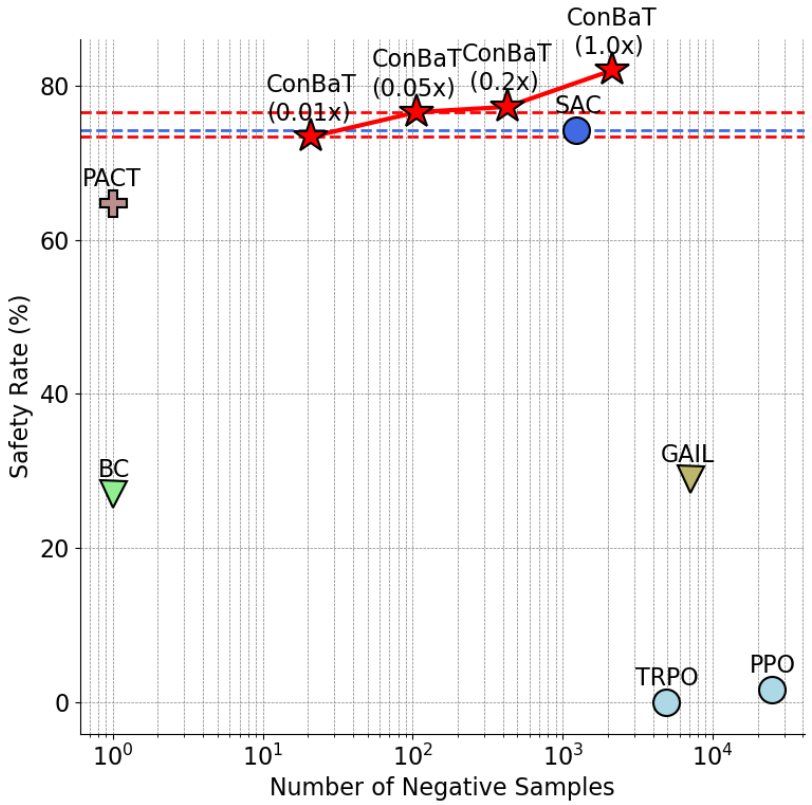


Figure 10: Rollout performance under different numbers of negative samples in training. ConBaT is the most efficient one that leverages negative samples: with only 21 negative examples, it can already learn the safety concept and improve the safety rate of based model (PACT) by 9%, and outperforms almost all the baselines.

C.7 Additional baselines

Besides the baselines shown in the main paper, we also compare with constrained policy optimization (CPO) (Achiam et al., 2017) and a model-base safe control approach called SABLAS (Qin et al., 2022). For CPO, we adapted the implementation from `https://github.com/SapanaChaudhary/PyTorch-CPO` as the official implementation is coded in Theano, which is incompatible with our simulation framework. For SABLAS, we followed the official implementation `https://github.com/MIT-REALM/sablas` and adapted it to our simulation environment. For CPO and SABLAS, the (safety) constraint is that the shortest lidar beam should be always

Algorithm	Training time (h/m)	Runtime (s)	Collision (%)	ATL (# steps)
CPO (Achiam et al., 2017)	12h	449.54	1.00	82.14
SABLAS (Qin et al., 2022)	12h	600.79	0.99	145.47
PACT	11h 2m	666.51	0.35	3453.34
ConBaT	12h 5m	852.31	0.18	4271.54

Table 7: ConBaT outperforms extra baselines CPO and SABLAS on safe navigation in the 2D MuSHr car domain.

greater than 0.1m. We trained the CPO for 10000 epochs using 12 hours and trained SABLAS for 20000 iterations using 12 hours. As shown in Table. 7, our approach outperforms the two baselines in both collision rate and average trajectory length. The inferior result of those two baselines might result from the extremely high dimension system dynamics and the high dimension constraint function (720x2=1440d), whereas our approach does not need to learn the explicit system dynamics or learn from the constraint on the explicit state space (we learn the implicit system dynamics and safety concept from the safe/unsafe demonstrations).

C.8 Optimality-MuShR experiment

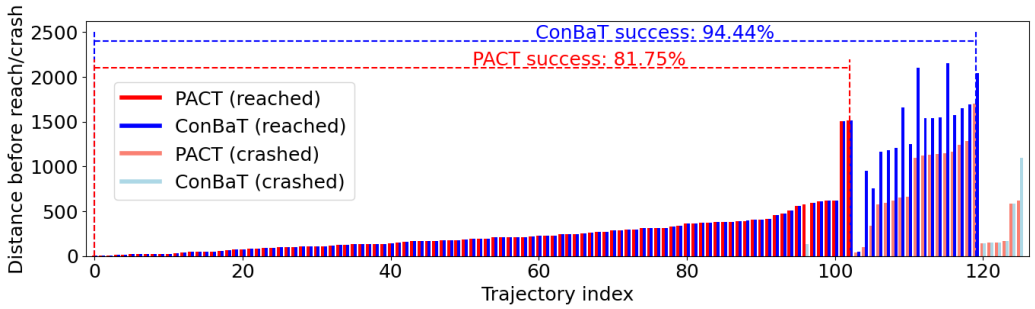


Figure 11: Trajectory comparison between PACT and ConBaT for the MuShR goal-reaching task. ConBaT results in higher goal-reaching success rate, and can preserve the same solution quality as the PACT model when the PACT trajectory is not crashed. ConBaT also results in 64% overhead for the average goal-reaching distance, which happens when the PACT trajectories crashes.

To investigate how ConBaT will affect the optimality of the solution, we design a goal-reaching task under the MuShR simulation environment, collect expert demonstrations and train PACT and ConBaT to reach the destination point. Specifically, the task is to reach a fixed destination point $(-9.2, -17.5)^T$ on the map from a randomly initialized position. We collect 10000 expert trajectories using the search-based MPC, which is the same one used in the previous MuShR data collection process in the main paper, but with a fixed destination point (and 10000 different initialized points).

We train the PACT and ConBaT on the expert data, following the same set of hyperparameters used in the main paper. During testing, we test for 128 different initial starting points, use the controller trained by PACT/ConBaT to rollout for at most 5000 time steps for evaluation. We define the success rate as the percentage of rollout trajectories that can reach the goal without any collision, compute the average trajectory length before reaching goal, and the average trajectory length before collision.

We plot the length of each trajectory before goal-reaching/collision for each method for comparison in Figure. 11. We categorize the trajectories depending on the "goal-reaching/crashing" consequence of the PACT/ConBaT trajectories, and inside each region we sort those trajectories based on the PACT trajectory length before crashing/goal-reaching. As shown in Figure. 11, ConBaT achieves a success rate of 94.44%, which is 12.69% higher than the PACT model. Our method doesn't improve the optimality/quality of each individual solution, because ConBaT is designed for improving safety rather than optimality. However, compared to PACT model, ConBaT maintains the quality of the solution (indices 0-102) when the PACT trajectories are safe and only increases the trajectory

length when the PACT trajectories are crashing (indices 103-127). This shows ConBaT can work in a shield-like fashion which preserves the base model behavior when safe, and only changes the trajectory when the potential unsafe case emerges.

C.9 Ablation study for CBF trend loss

One might think it is straight-forward to use safe/unsafe labels to guide the safe learning process. Here we emphasize the importance of the smoothness loss (trend loss). From the same PACT base model under MuSHr environment, we train the ConBaT to learn the safety score by differnet weighting for the smoothness term (ranging from 0-50, where in the main paper we use $\mathcal{L}_s = 5$). We follow the same optimization process and rollout mechanism in rollout. As shown in Table. 8, without the smoothness loss or when $\lambda_s \leq 2.0$, the ConBaT cannot improve upon the PACT performance. This makes sense as a smaller trend is not advance enough to prevent the states fall into unsafe region (but too large trend loss will hurt the CBF classification result hence affect the correctness for the CBF-critic to judge whether next state is really safe or unsafe). With large smoothness loss ($\lambda_s \geq 5.0$) added to the ConBaT training, the performance gets better than PACT and the peak is from $\lambda_s = 5$. Thus we picked $\lambda_s = 5$ in our experiments.

Method	Smoothness weight λ_s	Collision (%)	ATL (# steps)
PACT	-	0.35	3453.34
ConBaT	0	0.63	2799.79
ConBaT	0.1	0.88	2086.95
ConBaT	0.2	0.88	2004.23
ConBaT	0.5	0.76	2262.83
ConBaT	1	0.66	2991.55
ConBaT	2	0.91	2147.82
ConBaT	**5**	**0.18**	**4271.54**
ConBaT	10	0.23	3944.15
ConBaT	20	0.30	3654.66
ConBaT	50	0.35	3453.34

Table 8: ConBaT training under different smoothness weights.