Appendix 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 [58]. The racetrack is assumed to be 2 m wide, and the observation space for the agent is two dimensional, reporting the distance to the center line, 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 Fig. 4(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 length of the collected trajectories 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 Fig. 4(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. A heatmap showing the coverage of the map in the collected trajectories is shown in Figure 9

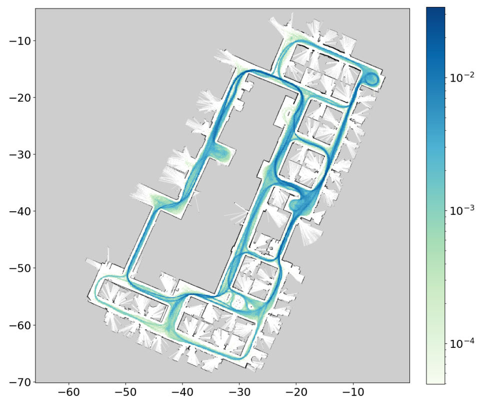


Figure 9: Visualization of the MuSHR environment occupancy map and collected data distribution

Appendix 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 the 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. We list the hyperparameters used for model training in Table 2.

Table 2: Hyperparameters

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

Deployment and 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 the prompt containing the past observations and actions, along with 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 rollout procedure of generating a new action and the corresponding 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 history of states and actions represented as the 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 an action proposal for time t. Through the online optimization step, our aim is to evaluate whether the proposed action needs to be modified further. 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:

$$\mathcal{L}_v = \sigma_+(\eta - C_f(s_t^+, a_t^+)) \tag{6}$$

where η is a parameter representing the safety threshold. We recall that a CBC value of 0 indicates a crash/failure. If $\eta > 0$, that means the CBC value should be greater than a positive value to satisfy the CBF condition, akin to asking the policy to be more conservative. We only perform online optimization if $\mathcal{L}_v$ is nonzero. For this routine, we use the RMSProp optimizer and do $1 \sim 3$ backward steps to compute Δa. We present a more detailed ablation study for several online optimization configurations 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 trained the Soft-Actor-Critic algorithm [62] 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 trained the TRPO algorithm [61] 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 trained the PPO algorithm [60] 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 trained the GAIL algorithm [29] 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. For simple behavior cloning, we trained 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.

CPO. 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. We trained the CPO algorithm for 10000 epochs which took approximately 12 hours.

SABLAS. 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 greater than 0.1m. We trained SABLAS for 20000 iterations over approximately 12 hours.

Appendix C Additional Results And Ablation Studies

C.1 Computational Effort

In Table 3, 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 reinforce-

Table 3: Training and deployment time taken

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

ment learning baselines are relatively faster in deployment than PACT or ConBaT, their training time is much higher.

C.2 Variations in critic architecture

As discussed in Section 3, we can use different network structures to function as the control barrier critic. We compare the following architectures:
CBC-NW: Uses both C and C_f, but no world model ϕ;
CBC-CW: Only uses the current state critic C, and the output of the world model is fed into it for the next state's cost.
CBC-TF: Uses C, C_f, and ϕ. The input to C_f is the output state embedding and the action input token (s_t^+, a_t').
CBC-EF: Uses C, C_f, and ϕ. The input to C_f is the output state embedding and the output action embedding (s_t^+, a_t^+).

We compare all methods in Table 4. We note that the world model is indeed useful, as evidenced by the lowest performance of CBC-NW. Using a future state critic results in slightly better performance than only using a current state critic that takes world model simulated states, but higher computational effort. Lastly, CBC-EF achieves higher performance than CBC-TF, indicating that the critic benefits from the context that plays a part in the computation of the action embedding, as opposed to the single token. We note that CBC-EF is used for all other ConBaT experiments in the paper.

Table 4: Comparison of architectures, F1 simulator

Arch.	Collision (%)	ATL (# steps)	Runtime (s)
PACT	100	175.45	63.328
CBC-NW	4.69	952.47	129.595
CBC-CW	1.56	983.48	200.153
CBC-TF	3.91	972.42	78.159
CBC-EF	0.00	1000	130.032

C.3 Data Requirements

The existence of demonstrations that include unsafe terminal states is critical for ConBaT to be able to encode the safety constraints. Yet, given that simulating and collecting a large amount of unsafe demonstrations can be challenging (and costly in real world conditions), we perform a analysis of how ConBaT performs in low negative data regimes when training the critic (phase II). Table 5 reports performance levels for varying unsafe dataset sizes. We observe that ConBaT can learn safe navigation even with small amounts of unsafe demonstrations.

We perform a more extensive analysis of data requirements in the MuSHR domain. We compare the performance of RL and IL baselines with ConBaT along with how many unsafe trials were utilized in the training process. We evaluate the percentage of safe trajectories executed across 128

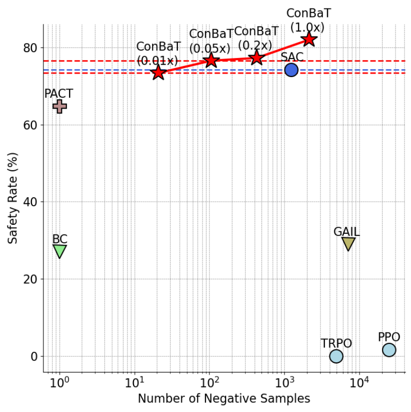


Figure 10: Roll-out performance of policies under different numbers of negative samples (terminal crashes) observed in the training phase. ConBaT is the most efficient method in leveraging negative experiences: with only 21 negative examples, it can already learn the safety concept and improve the safety rate of based model (PACT) by 9%, and outperforms almost all the baselines.

initial conditions with each trajectory capped at a maximum of 5000 timesteps. For ConBaT, we use different sizes of unsafe trials (ranging from 21 examples, which is 1% of the full unsafe dataset to 2131, which is 100%). As shown in Fig. 10, 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 just a few more (106) negative examples, we already surpass all the baselines considered in this paper. This again showcases the data-efficiency of our algorithm, contrasting it with the high number of unsafe trials usually required by RL.

Table 5: Amount of unsafe trajectories used in phase II training

% Trajs	Collision (%)	ATL (# steps)	Runtime (s)
0	100	175.45	63.328
5	5.47	944.67	120.427
10	3.91	960.18	132.079
25	1.56	983.47	130.049
50	3.91	960.26	127.275
100	0.78	991.22	127.073

C.4 CBF Critic Ablations

In Table 6, we compare two different architecture designs for the future state critic C_f. CBC-EF (stands for CBC-embedding/future) which takes the current state and action embeddings as input: $\hat{c}_{t+1} = C_f(s_t^+, a_t^+)$, and CBC-TF (CBC-token/future) which takes the current state embedding but only the current action token as input: $\hat{c}_{t+1} = C_f(s_t^+, a_t')$. As shown in Table 6, the online optimization time for CBC-TF is $40\% \sim 60\%$ shorter than the runtime of CBC-EF, which is because in CBC-TF the action token does not have to be processed through the Transformer and temporally-fused with features from other timesteps. However, we find that consistently among all the three tracks, CBC-EF achieves much better collision rate and ATL compared to CBC-TF. We attribute to the feature richness of the action embeddings after fusing with other state-action history, compared to the raw action tokens. Thus, we primarily use the CBC-EF architecture in our paper.

C.5 Online Optimization Ablations

Table 7 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 ex-

Table 6: Using action embedding vs. action token for the Control Barrier Critic

Track	Collision Rate (%)		ATL (# steps)		Runtime (sec)	
	CBC-EF	CBC-TF	CBC-EF	CBC-TF	CBC-EF	CBC-TF
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

amine the effects of these three parameters: a) the number of gradient descent steps, b) the learning rate, and c) the CBF safety threshold η. From Table 7(a), we note that increasing the optimization steps in the online optimization does not improve the performance. From Table 7(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 7(c), we see that the threshold η to some degree reflects the conservativeness - a larger threshold will result in a more conservative policy, which potentially can lead to better performance.

(a) SGD step.

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

(b) Learning rate (lr)

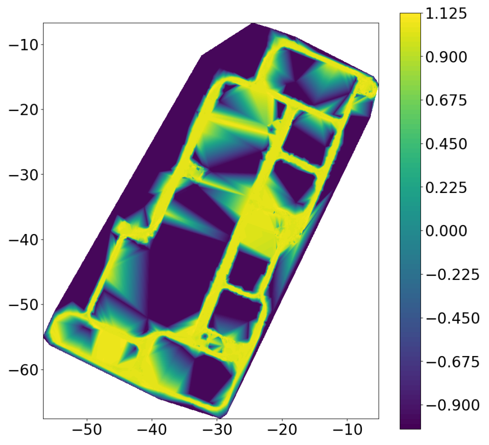
lr	Collision	ATL
0.05	0	1000
0.1	0.0078	991.23
0.2	0.0078	991.22
0.3	0.0156	983.4
0.5	0.0234	975.73

(c) Threshold

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

Table 7: Ablation studies for online optimization (iterations, learning rates and thresholds)

C.6 Learned Control Barrier Critic Visualization



(a) Visualization from Control Barrier Critics (CBC)

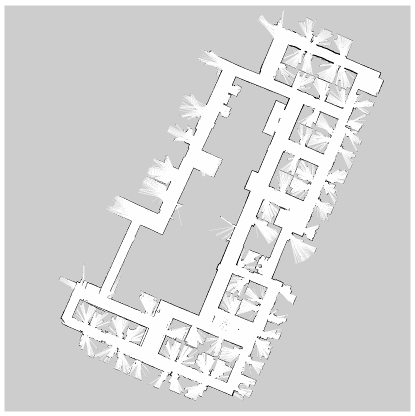

(b) Visualization of the original MuSHR environment

Figure 11: Visualization of Control Barrier Critics for MuSHR

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 Figure 11a, 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.7 Trade-off between safety and optimality -MuSHR domain

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 12. 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 12, 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.

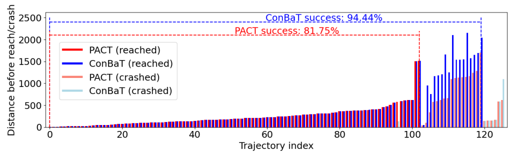


Figure 12: Trajectory comparison between PACT and ConBaT for the MuSHR goal-reaching task. ConBaT results in higher goal-reaching success rate, and can preserve the same solution quality as the PACT model when the PACT trajectory is not crashed. ConBaT also results in 64% overhead for the average goal-reaching distance, which happens when the PACT trajectories crashes.

C.8 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

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.

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.

C.9 Different architectures for safe policy learning on MuSHR

Architecture	Params.	Training	Runtime (s)	Collision (%)	ATL
MLP	**243k**	**8h 43m**	**351.47**	34.38	3580.01
MLP+CBC	293k	9h 47m	419.42	28.91	3721.06
LSTM	343k	9h 49m	383.94	53.12	2652.30
LSTM+CBC	393k	10h 54m	425.09	47.66	2802.03
GRU	376k	8h 51m	373.15	35.16	3468.57
GRU+CBC	426k	9h 56m	382.02	31.25	3613.84
PACT	310k	11h 2m	383.32	35.16	3453.34
PACT+CBC	360k	12h 5m	411.34	**17.97**	**4271.54**

Table 9: Policy learning on MuSHR dataset under varied architectures and optimization options (CBC). The runtime is different from Table 3 because of the different GPU resources used here.

To study the importance of the Transformer backbone in safe policy learning, we replace it with different architectures and repeat the same training, fine-tuning, and test procedures as the ConBaT. To be more specific, we consider three other architectures, Multi-layer Perception (MLP), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU). For MLP, for each time step, we simply concatenate the current input tokens with all its history input tokens (pad with zero if none) with the 16-step time horizon and send them to the MLP backbone (2 hidden layers with 128 neurons in each layer) to generate the output embeddings. For LSTM and GRU, we use two recurrent layers with a hidden state size of 128; we have a projection layer to cast the final output (which has the same dimension as the hidden states) to the output embedding space. For each backbone, we record the performance of the model trained in the first stage (without CBC learning or optimization, denote as "Backbone Name") as well as after the second stage (with CBC learning and optimization, similar to ConBaT, denote as "Backbone Name+CBC."). To make a fair comparison, we do a grid search for each architecture over 50 configurations for the online optimization phase and report the best one. As shown in Table 9, in the first stage, the MLP backbone achieves the lowest collision rate and the highest ATL, slightly better than GRU and transformer backbones. And in the second stage, all the backbones can achieve $4\% \sim 17\%$ collision rate improvement after the optimization process, which indicates that ConBaT is architecture-agnostic. Among all the architectures, the Transformer backbone achieves the most significant reduction in the collision rate (17%) and achieves the lowest collision rate. This is likely because the safety-ness in MuSHR depends on both the current state and the history state/action information, and the Transformer backbone excels at modeling the long-term dependencies. Hence the CBC can better tell the safety scores of the states, and correspondingly, the controller can achieve the best performance.