Safe Offline Reinforcement Learning via Chance-Constrained Policy Filtering

Supplementary Material

IROS 2026

Kangyeon Kim, Seonho Yoo, and Heejin Ahn

I. Details for Posterior Computation and Calibration

A. Prior Setting

We set the prior probabilities externally and use them to compute the conservative posterior in Equation (7) of the main text:

$$\mathbb{P}^{\xi}_{D_{\text{off}}}[c=k \mid \hat{c}=j] := \frac{\mathbb{P}^{+\xi}_{D_{\text{off}}}[\hat{c}=j \mid c=k]\cdot \text{prior}(c=k)}{\sum_{q=0}^{1}\mathbb{P}^{-\xi}_{D_{\text{off}}}[\hat{c}=j \mid c=q]\cdot \text{prior}(c=q)}, \tag{1}$$

where $\mathbb{P}^{+\xi}_{D_{\text{off}}}[\hat{c}=j \mid c=q] := \frac{N^{+\xi}_{D_{\text{off}}}(c=q,\hat{c}=j)}{N_{D_{\text{off}}}(c=q)}$ and $\mathbb{P}^{-\xi}_{D_{\text{off}}}[\hat{c}=j \mid c=q] := \frac{N^{-\xi}_{D_{\text{off}}}(c=q,\hat{c}=j)}{N_{D_{\text{off}}}(c=q)}$. Although the empirical cost frequency in D_{off} provides a natural choice of prior, it primarily reflects the data collection distribution. In contrast, the filtered policy is constructed from the base policy π_{off}, whose violation tendency may vary across algorithm–task combinations. To incorporate this policy-dependent variation into the posterior computation, we use the average cost rate of π_{off} as a practical proxy for the external violation prior.

We assume access to an estimate of the average episodic cost of the base policy π_{off}, obtained through evaluation. We convert this estimate into an average cost rate by dividing it by the episode length. For each algorithm–task combination, we evaluate π_{off} using three random seeds and define C_{off} as the minimum of the resulting average cost rates to avoid an excessively conservative prior caused by variability across random seeds. We then set

$$\big(\text{prior}(c=0), \text{prior}(c=1)\big) = (1-C_{\text{off}}, C_{\text{off}}).$$

The detailed values of C_{off}, together with the corresponding ranges of average cost rates across random seeds, are summarized in Table I. Intuitively, this prior reflects how conservatively or aggressively the base policy behaves.

TABLE I
Prior Parameter C_{off}

BCQ	CarGoal1	CarGoal2	PointGoal1	PointGoal2
Average cost rate (min, max)	0.018, 0.033	0.046, 0.058	0.033, 0.047	0.085, 0.111
C_{off}	0.018	0.046	0.033	0.085
BEAR	CarGoal1	CarGoal2	PointGoal1	PointGoal2
Average cost rate (min, max)	0.028, 0.043	0.075, 0.095	0.028, 0.036	0.081, 0.100
C_{off}	0.028	0.075	0.028	0.081

II. Details for Back-propagation

While calibrating the cost estimator guarantees satisfaction of the chance constraint, it can be overly conservative and may degrade performance. In particular, the bias-correction step shifts the decision boundary solely to enforce the posterior constraint without explicitly accounting for reward optimization.

To alleviate this conservativeness, we further fine-tune the cost estimator via back-propagation while keeping the base policy π_{off} fixed. The key idea is to adapt the decision boundary in a manner guided by the base policy π_{off}, thereby accounting for performance in addition to safety. When two candidate actions yield similar (or identical) logits, they may be assigned to the same class and thus receive the same estimated violation probabilities. In such cases, bias correction alone may not distinguish between them; however, it is preferable to select the action that is more likely under π_{off}, since π_{off} is trained to maximize reward.

The authors are with the School of Electrical Engineering, KAIST, Daejeon, Republic of Korea. ydk1235@naver.com, sh.yoo@kaist.ac.kr, heejin.ahn@kaist.ac.kr

Project Page: https://safeofflinerl-cpf.github.io

A. Definition of J

To define J, we partition the action space A into G representative groups of behaviorally similar actions, $A^{(i)}$ for $i \in \{1,\dots,G\}$. Define the group index map $h : A \to \{1,\dots,G\}$ by $\mathbf{a} \in A^{(h(\mathbf{a}))}$. We then define $J(\mathbf{a};\mathbf{s})$ to reflect π_{off}:

$$J(\mathbf{a};\mathbf{s}) = \begin{cases} 0, & \text{if } \max_{\mathbf{a}\in A} P(\mathbf{a};\mathbf{s}) - \min_{\mathbf{a}\in A} P(\mathbf{a};\mathbf{s}) = 0, \\ -P(\mathbf{a};\mathbf{s}), & \text{else if } \max_{\mathbf{a}\in A} y(\mathbf{a};\mathbf{s}) - \min_{\mathbf{a}\in A} y(\mathbf{a};\mathbf{s}) = 0, \\ -R(\mathbf{s}) \cdot P(\mathbf{a};\mathbf{s}), & \text{otherwise.} \end{cases}$$

where

$$P(\mathbf{a};\mathbf{s}) := \frac{\int_{\mathbf{a}'\in A^{(h(\mathbf{a}))}} \pi_{\text{off}}(\mathbf{a}' \mid \mathbf{s})\, d\mathbf{a}'}{\int_{\mathbf{a}'\in A} \pi_{\text{off}}(\mathbf{a}' \mid \mathbf{s})\, d\mathbf{a}'},$$

$$y(\mathbf{a};\mathbf{s}) := -\beta \min\big(\delta' - \mathbb{P}^{\xi}_{D_{\text{off}}}[c = 1 \mid \hat{c} = j], 0\big)$$

$$R(\mathbf{s}) := w \cdot \frac{\max_{\mathbf{a}\in A} y(\mathbf{a};\mathbf{s}) - \min_{\mathbf{a}\in A} y(\mathbf{a};\mathbf{s})}{\max_{\mathbf{a}\in A} P(\mathbf{a};\mathbf{s}) - \min_{\mathbf{a}\in A} P(\mathbf{a};\mathbf{s})},$$

and $j = \hat{c}(\mathbf{s},\mathbf{a})$. Here, w is a hyperparameter that controls the relative weighting between policy preference and constraint violation. First, $P(\mathbf{a};\mathbf{s})$ is the normalized probability mass that $\pi_{\text{off}}(\cdot \mid \mathbf{s})$ assigns to the group $h(\mathbf{a})$. If $P(\mathbf{a};\mathbf{s})$ is constant over $\mathbf{a} \in A$ (i.e., $\max_{\mathbf{a}\in A} P(\mathbf{a};\mathbf{s}) - \min_{\mathbf{a}\in A} P(\mathbf{a};\mathbf{s}) = 0$), then π_{off} does not prefer any candidate, and we set $J(\mathbf{a};\mathbf{s}) = 0$. Second, $y(\mathbf{a};\mathbf{s})$ penalizes constraint violation and increases when an action violates the chance constraint. If the penalty term does not vary across candidates (i.e., $\max_{\mathbf{a}\in A} y(\mathbf{a};\mathbf{s}) - \min_{\mathbf{a}\in A} y(\mathbf{a};\mathbf{s}) = 0$), then safety does not distinguish candidates, and selection should follow the policy preference; hence we set $J(\mathbf{a};\mathbf{s}) = -P(\mathbf{a};\mathbf{s})$. Otherwise, we use the scaling term $R(\mathbf{s})$ to balance the policy-preference and violation-penalty terms, preventing either from dominating the surrogate objective.

In the approximated general loss function

$$\tilde{L}(j) = \min_{\mathbf{a}\in A}\Big(J(\mathbf{a};\mathbf{s}) - \beta \min\big(\delta' - \mathbb{P}^{\xi}_{D_{\text{off}}}[c = 1 \mid \hat{c} = j], 0\big)\Big),$$

we set the parameter β as follows:

$$\beta := \beta'\Big(1 + \alpha \cdot \mathbb{I}\{c(\mathbf{s},\mathbf{a}) = 1,\ j = 0\}\Big),$$

where

$$\alpha = \begin{cases} 0, & \text{if } \max_j x(j) = 0, \\ \left|\frac{2\beta'\left(\max_j x(j) - \min_j x(j)\right)}{\max_j x(j)}\right|, & \text{otherwise,} \end{cases}$$

$$x(j) := \min\big(\delta' - \mathbb{P}^{\xi}_{D_{\text{off}}}[c = 1 \mid \hat{c} = j], 0\big),$$

and β' is a hyperparameter. This parameterization increases the penalty weight when the selected action is observed to be unsafe (i.e., $c(\mathbf{s},\mathbf{a}) = 1$) while the cost estimator assigns it to the predicted-safe. In other words, β amplifies the gradient signal for such false-negative cases by utilizing information from the offline dataset. In detail, we use $w = 1.4$ and $\beta' = 1.5$.

B. Action Space Partitioning into G Groups

To define $J(\mathbf{a};\mathbf{s})$, we partition the continuous action space into a finite number of representative groups. The grouping is designed to cluster behaviorally similar actions together.

For the `Point` robot, the action space is $[-1,1]^2$. Since the effect of actions varies smoothly across this range, we apply a uniform discretization. Each action dimension is divided into equal intervals with a step size of 0.20, resulting in a regular grid over the action space. Actions that fall within the same grid cell are assigned to the same group, yielding $G = 100$ groups.

For the `Car` robot, although the action space is also $[-1,1]^2$, empirical observations show that small-magnitude forces around zero (approximately within $[-0.05, 0.05]$) produce behavior nearly identical to zero input, while forces outside this interval tend to induce motion patterns qualitatively similar to those produced by strongly positive or negative inputs. Accordingly, instead of uniform discretization, we partition each action dimension into three regions corresponding to negative, near-zero, and positive inputs. This results in $G = 9$ groups in total. This non-uniform partition better reflects the effective dynamics of the car and groups actions according to their behavioral impact.

III. Details for Application

A. Setting of the Default Action $\mathbf{a}_{def}$

When the state $\mathbf{s}$ is infeasible, i.e., $\bar{A}_{\text{safe}}(\mathbf{s}) = \varnothing$, we execute a default action $\mathbf{a}_{\text{def}}$. The default action is determined based on the current translational velocity component of $\mathbf{s}$.

Specifically, we use the sign of the velocity to infer the current motion direction. If the agent is moving forward, we apply a backward default action; if it is moving backward, we apply a forward default action. When the velocity magnitude is small (i.e., near zero), we apply a stop action.

The concrete default actions are robot-specific. For the `Car` robot, forward, backward, and stop actions correspond to $[1,1]$, $[-1,-1]$, and $[0,0]$, respectively. For the `Point` robot, forward and backward actions correspond to $[-1,0]$ and $[1,0]$, respectively, while the stop action remains $[0,0]$.

Lastly, repeatedly executing the default action after entering a hazardous region may cause the robot to get stuck. To prevent this, we simply deploy the original policy π_{off} at the next step whenever the previous state is unsafe.

B. BEAR Evaluation Procedure

In BEAR, the actor outputs the mean and standard deviation of a Gaussian distribution and samples actions from this distribution. Thus, when we use BEAR as π_{off}, we discretize a bounded region around the mean to construct $\bar{A}_{\text{cand}}(\mathbf{s})$, thereby obtaining action candidates that sufficiently cover the probability mass of $\pi_{\text{off}}(\cdot \mid \mathbf{s})$. In our experiments, the action space is two-dimensional. We fix the standard deviation to $[0.5, 0.5]$ to avoid overly narrow candidate coverage. This empirically reduces the frequency of infeasible cases (i.e., $\bar{A}_{\text{safe}}(\mathbf{s}) = \varnothing$) without significantly degrading performance.

Algorithm 1 Evaluation for BEAR

(a) Original evaluation for BEAR

Require: Actor π_{off}, Environment `env`

Get the initial state: $\mathbf{s}_1 \leftarrow$ `env.reset()`
for $t = 1$ to T **do**
  $N(\mu_t, \sigma_t) \leftarrow \pi_{\text{off}}(\cdot \mid \mathbf{s}_t)$
  $\mathbf{a}_t \sim N(\mu_t, \sigma_t)$ or $\mathbf{a}_t = \mu_t$
  $\mathbf{a}_t \leftarrow$ `tanh(`$\mathbf{a}_t$`)`
  Execute the action: $\mathbf{s}_{t+1}, r_t, c_t \leftarrow$ `env.step(`$\mathbf{a}_t$`)`
end for

(b) Evaluation for BEAR+Ours

Require: Policy π_{off}, Environment `env`, Cost estimator $\hat{c}$

Get the initial state: $\mathbf{s}_1 \leftarrow$ `env.reset()`
for $t = 1$ to T **do**
  $N(\mu_t, \sigma_t) \leftarrow \pi_{\text{off}}(\cdot \mid \mathbf{s}_t)$
  Construct $\bar{A}_{\text{cand}}(\mathbf{s}_t)$ by discretizing a bounded box around μ_t
  $\bar{A}_{\text{safe}}(\mathbf{s}_t) \leftarrow \{\bar{\mathbf{a}} \in \bar{A}_{\text{cand}}(\mathbf{s}_t) \mid \hat{c}(\mathbf{s}_t, \bar{\mathbf{a}}) = 0\}$
  $\bar{\mathbf{a}}_t \sim$ Renormalize $\pi_{\text{off}}(\cdot|\mathbf{s}_t)$ over $\bar{A}_{\text{safe}}(\mathbf{s}_t)$
  $\bar{\mathbf{a}}_t \leftarrow$ `tanh(`$\bar{\mathbf{a}}_t$`)`
  Execute the action: $\mathbf{s}_{t+1}, r_t, c_t \leftarrow$ `env.step(`$\bar{\mathbf{a}}_t$`)`
end for

C. BCQ Evaluation Procedure

In contrast, BCQ generates actions by sampling a latent vector $\mathbf{z}$ and decoding it through a variational autoencoder (VAE), followed by perturbation. Thus, instead of discretizing actions directly, we discretize the latent space, decode it into actions via the VAE, and apply perturbation to obtain $\bar{A}_{\text{cand}}(\mathbf{s})$. Similar to BEAR, we discretize a larger latent range to obtain a more diverse set of action candidates, thereby reducing the frequency of infeasible cases.

Algorithm 2 Evaluation for BCQ

(c) Original evaluation for BCQ

Require: Perturbation model ξ_ϕ, VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, Environment `env`

Get the initial state: $\mathbf{s}_1 \leftarrow$ `env.reset()`
for $t = 1$ to T **do**
  $\mathbf{z}_t \sim N(\mathbf{0}, \mathbf{I}_4)$
  Clamp $\mathbf{z}_t$ element-wise to $[-0.5,\ 0.5]$
  $\tilde{\mathbf{a}}_t \leftarrow D_{\omega_2}(\mathbf{s}_t, \mathbf{z}_t)$
  $\mathbf{a}_t \leftarrow \tilde{\mathbf{a}}_t + \xi_\phi(\mathbf{s}_t, \tilde{\mathbf{a}}_t, \Phi)$
  Execute the action: $\mathbf{s}_{t+1}, r_t, c_t \leftarrow$ `env.step`$(\mathbf{a}_t)$
end for

(d) Evaluation for BCQ+Ours

Require: Perturbation network ξ_ϕ, VAE $G_\omega = \{E_{\omega_1}, D_{\omega_2}\}$, Environment `env`, Cost estimator $\hat{c}$

Construct $\bar{Z}$ by discretizing $[-1.0,\ 1.0]^4$ into a grid
Get the initial state: $\mathbf{s}_1 \leftarrow$ `env.reset()`
for $t = 1$ to T **do**
  $\bar{A} \leftarrow D_{\omega_2}(\mathbf{s}_t, \bar{Z})$
  $\bar{A}_{\text{cand}}(\mathbf{s}_t) \leftarrow \bar{A} + \xi_\phi(\mathbf{s}_t, \bar{A}, \Phi)$
  $\bar{A}_{\text{safe}}(\mathbf{s}_t) \leftarrow \{\bar{\mathbf{a}} \in \bar{A}_{\text{cand}}(\mathbf{s}_t) \mid \hat{c}(\mathbf{s}_t, \bar{\mathbf{a}}) = 0\}$
  $\bar{\mathbf{a}}_t \sim$ Renormalize the action distribution induced by $\mathbf{z} \sim N(\mathbf{0}, \mathbf{I}_4)$ over $\bar{A}_{\text{safe}}(\mathbf{s}_t)$
  Execute the action: $\mathbf{s}_{t+1}, r_t, c_t \leftarrow$ `env.step`$(\bar{\mathbf{a}}_t)$
end for

IV. Details for Selection of Baselines

For baseline comparison, we select BEARL, BCQL, CAPS, and CDT. First, our method is designed as a plug-in module that can be integrated with various offline RL algorithms, while also supporting adaptation to different safety thresholds at deployment without retraining. Constraint-Adaptive Policy Switching (CAPS) shares this deployment-time threshold adaptivity and can be combined with existing offline RL backbones (e.g., SAC or IQL). We therefore use CAPS as a primary baseline that matches the core functionality of our approach.

Second, we include CDT, which formulates safe offline RL as threshold-conditioned sequence modeling and demonstrates zero-shot generalization across different safety thresholds. Although CDT cannot be directly integrated with existing offline RL algorithms as a plug-in module, it addresses the same goal of handling varying safety thresholds at deployment.

Lastly, we adopt conventional Lagrangian-based constrained optimization baselines (BEARL and BCQL), which incorporate safety constraints into BEAR and BCQ via a Lagrangian multiplier. While these methods typically require separate training or tuning for each target threshold, they represent the standard approach for enforcing constraints in safe offline RL and provide a strong reference point for comparison.

V. Details for Cost Estimator

A. Cost Estimator Architecture and Training Hyperparameters

We pretrain the cost estimator using the loss function

$$L = L_{\text{reg}} + \alpha_L L_{\text{bce}},$$

where

$$L_{\text{reg}} = \left\| \mathbf{z}_\phi(\mathbf{s}_t^i, \mathbf{a}_t^i) - \mathbf{z}(\mathbf{s}_{t+1}^i) \right\|_2^2,$$

$$L_{\text{bce}} = \sum_{h=1}^{n_h} \text{BCEWithLogits}\left(z_\phi^h(\mathbf{s}_t^i, \mathbf{a}_t^i), \mathbb{I}\{z^h(\mathbf{s}_{t+1}^i) > 0\} \right).$$

In detail, we use $\alpha_L = 0.4$ and a batch size of 512. During evaluation, the classifier output is obtained via $g = \arg\max$. The detailed architecture and training hyperparameters are summarized in Table II.

B. Details for Ours(PT)

All methods are trained up to a task-specific budget $T_{\text{total}} \in \{40, 60, 80\} \times 10^4$ epochs, with checkpoints evaluated every 10^4 epochs. For the baselines (BCQL, BEARL, CDT, CAPS), we follow a per-seed model selection strategy: for each random seed, we select the checkpoint achieving the highest reward return, and then report the mean and standard deviation across three seeds. The results in Table I in the main text are computed using this procedure.

TABLE II
COST ESTIMATOR ARCHITECTURE AND TRAINING HYPERPARAMETERS

Section	Implementation	Parameter	Value
Shared Layer 1	`torch.nn.Linear`	`in_features`	`state_dim + action_dim`
		`out_features`	256
Shared Layer 1 activation	`torch.nn.ReLU`	–	–
Shared Layer 2	`torch.nn.Linear`	`in_features`	256
		`out_features`	256
Shared Layer 2 activation	`torch.nn.ReLU`	–	–
Shared Layer 3	`torch.nn.Linear`	`in_features`	256
		`out_features`	128
Shared Layer 3 activation	`torch.nn.ReLU`	–	–
Head Layer 1 (per head)	`torch.nn.Linear`	`in_features`	128
		`out_features`	128
Head Layer 1 activation (per head)	`torch.nn.ReLU`	–	–
Head Layer 2 (per head)	`torch.nn.Linear`	`in_features`	128
		`out_features`	1
Optimizer	`torch.optim.AdamW`	`lr`	10^{-3}
		`weight_decay`	10^{-6}
		`amsgrad`	`True`

For the base policies (BCQ and BEAR), we instead select a single checkpoint based on the highest reward averaged over three seeds. This selected checkpoint at epoch T_{base} serves as the starting point for cost-estimator pretraining and subsequent deployment experiments.

For each random seed s, we initialize the base policy at T_{base} and pretrain a cost estimator for the remaining budget of $T_{\text{total}} - T_{\text{base}}$ epochs. At each cost-estimator checkpoint, we evaluate the deployment performance obtained by combining the fixed base policy with the current cost estimator. For each seed, we then select the cost-estimator checkpoint that achieves the highest reward over 20 evaluation episodes. We denote the selected cost-estimator epoch for seed s and target threshold δ by $T_{\text{cost}}^{(s,\delta)}$, which may differ across seeds and thresholds. For each target threshold δ, the results reported as Ours(PT) in Table I of the main text are computed by averaging the corresponding reward and cost returns across seeds $s \in \{0, 10, 20\}$ using the selected checkpoints $\{T_{\text{cost}}^{(s,\delta)}\}_{s\in\{0,10,20\}}$. A summary of T_{total} and T_{base} is provided in Table III.

C. Details for Ours(BP)

The back-propagation experiment is conducted only for BEAR-based policies. For each random seed s and target threshold δ, we initialize the cost estimator from the previously selected checkpoint $T_{\text{cost}}^{(s,\delta)}$ obtained in Ours(PT). (The selected epochs $\{T_{\text{cost}}^{(s,\delta)}\}$ are summarized in Table III.) We then fine-tune the estimator for an additional 100×10^3 epochs, saving checkpoints every 2×10^3 epochs.

At each fine-tuning checkpoint, we evaluate the deployment performance obtained by combining the fixed base policy (at T_{base}) with the updated cost estimator. For each seed, we select the fine-tuned checkpoint achieving the highest reward over the evaluated checkpoints. The results reported as Ours(BP) in Table I in the main text are computed by averaging the reward and cost returns corresponding to the selected fine-tuned checkpoints across seeds.

TABLE III
CHECKPOINT SELECTION SETTINGS FOR BASE POLICIES AND COST ESTIMATORS (T_{TOTAL}, T_{BASE}, AND $T_{\text{COST}}^{(s,\delta)}$)

BCQ	`CarGoal1`	`CarGoal2`	`PointGoal1`	`PointGoal2`
T_{total}	40×10^4	80×10^4	60×10^4	60×10^4
T_{base}	24×10^4	21×10^4	22×10^4	48×10^4
T_{total} - T_{base}	16×10^4	59×10^4	38×10^4	12×10^4
BEAR	`CarGoal1`	`CarGoal2`	`PointGoal1`	`PointGoal2`
T_{total}	40×10^4	80×10^4	60×10^4	60×10^4
T_{base}	31×10^4	46×10^4	49×10^4	52×10^4
T_{total} - T_{base}	9×10^4	34×10^4	11×10^4	8×10^4
$T_{\text{cost}}^{(0,0.02)}$	7×10^4	14×10^4	7×10^4	4×10^4
$T_{\text{cost}}^{(10,0.02)}$	4×10^4	30×10^4	11×10^4	6×10^4
$T_{\text{cost}}^{(20,0.02)}$	8×10^4	30×10^4	11×10^4	5×10^4
$T_{\text{cost}}^{(0,0.04)}$	9×10^4	19×10^4	11×10^4	4×10^4
$T_{\text{cost}}^{(10,0.04)}$	4×10^4	33×10^4	9×10^4	5×10^4
$T_{\text{cost}}^{(20,0.04)}$	5×10^4	34×10^4	4×10^4	3×10^4
$T_{\text{cost}}^{(0,0.08)}$	1×10^4	30×10^4	6×10^4	4×10^4
$T_{\text{cost}}^{(10,0.08)}$	1×10^4	30×10^4	6×10^4	5×10^4
$T_{\text{cost}}^{(20,0.08)}$	1×10^4	22×10^4	10×10^4	4×10^4

VI. Details for Inaccurate Cost Estimator Experiments

To investigate the robustness of our method under imperfect cost estimation, we intentionally limit the pretraining of the cost estimator to 30×10^3 epochs. Checkpoints are saved every 2×10^3 epochs, yielding 15 estimator checkpoints per task. These early-stage checkpoints include estimators with relatively low recall.

To estimate the recall of each cost estimator, we collect 60×10^4 transitions per task using the fixed base policies (BEAR and BCQ). The resulting average recall is 0.39 on `CarGoal1` and 0.73 on `CarGoal2`.

For each cost-estimator checkpoint, we first average the 20-episode returns across the three seeds and then select the checkpoint with the highest average reward. As shown in Fig. 2 of the main text, all evaluated estimator checkpoints satisfy the safety constraint. When the estimator has lower recall, the deployed policies operate well below the safety threshold, reflecting increased conservatism induced by the filtering rule.

We additionally highlight the best checkpoints, which correspond to higher recall (average recall 0.75 on `CarGoal1` and 0.93 on `CarGoal2`). These checkpoints require a smaller conservatism margin and therefore achieve higher reward while satisfying the safety constraint.