A Proof

Proof of Proposition 4.2

Proposition 4.2 The performance gap of evaluating policy profile (π, μ) and (π, π_B) at state s is $\varepsilon(s) = V^{\pi,\pi_B}(s) - V^{\pi,\mu}(s)$, which can be decomposed in a Bellman-like recursion:

$$\begin{aligned}\varepsilon(s) &= V^{\pi,\pi_B}(s) - V^{\pi,\mu}(s) \\ &= \sum_{a_1,a_2,s'} \pi(a_1|s) P_M(s'|s,\boldsymbol{a}) \left(\pi_B(a_2|s) - \mu(a_2|s)\right) \left(r(s,\boldsymbol{a},s') + \gamma V^{\pi,\mu}(s')\right) \\ &\quad + \sum_{a_1,a_2,s'} \pi(a_1|s)\pi_B(a_2|s) P_M(s'|s,\boldsymbol{a}) \gamma \varepsilon(s')\end{aligned} \tag{7}$$

Proof.

$$\begin{aligned}\varepsilon(s) &= V^{\pi,\pi_B}(s) - V^{\pi,\mu}(s) \\ &= \sum_{a_1} \pi(a_1|s) \sum_{a_2} \pi_B(a_2|s) \sum_{s'} P_M(s'|s,\boldsymbol{a}) \left[r(s,\boldsymbol{a},s') + \gamma V^{\pi,\mu}(s') + \gamma\varepsilon(s')\right] \\ &\quad - \sum_{a_1} \pi(a_1|s) \sum_{a_2} \mu(a_2|s) \sum_{s'} P_M(s'|s,\boldsymbol{a}) \left[r(s,\boldsymbol{a},s') + \gamma V^{\pi,\mu}(s')\right] \\ &= \sum_{a_1} \pi(a_1|s) \sum_{a_2} \left[\left(\pi_B(a_2|s) - \mu(a_2|s)\right) A + B\right],\end{aligned} \tag{8}$$

where

$$A = \sum_{s'} P_M(s'|s,\boldsymbol{a}) r(s,\boldsymbol{a},s'), \tag{9}$$

and

$$\begin{aligned}B &= \pi_B(a_2|s) \sum_{s'} P_M(s'|s,\boldsymbol{a}) \gamma \left(V^{\pi,\mu}(s') + \varepsilon(s')\right) - \mu(a_2|s) \sum_{s'} P_M(s'|s,\boldsymbol{a}) \gamma V^{\pi,\mu}(s') \\ &= \sum_{s'} P_M(s'|s,\boldsymbol{a}) \left[\pi_B(a_2|s) \gamma \left(V^{\pi,\mu}(s') + \varepsilon(s')\right) - \mu(a_2|s) \gamma V^{\pi,\mu}(s')\right] \\ &= \sum_{s'} P_M(s'|s,\boldsymbol{a}) \left[\gamma V^{\pi,\mu}(s')(\pi_B(a_2|s) - \mu(a_2|s)) + \gamma \pi_B(a_2|s)\varepsilon(s')\right].\end{aligned} \tag{10}$$

Putting equation 9 and 10 back, we have

$$\begin{aligned}\varepsilon(s) &= V^{\pi,\pi_B}(s) - V^{\pi,\mu}(s) \\ &= \sum_{a_1} \pi(a_1|s) \sum_{a_2} \Bigg[\left(\pi_B(a_2|s) - \mu(a_2|s)\right) \sum_{s'} P_M(s'|s,\boldsymbol{a}) r(s,\boldsymbol{a},s') + \\ &\qquad \sum_{s'} P_M(s'|s,\boldsymbol{a}) \left[\gamma V^{\pi,\mu}(s')(\pi_B(a_2|s) - \mu(a_2|s)) + \gamma \pi_B(a_2|s)\varepsilon(s')\right] \Bigg] \\ &= \sum_{a_1,a_2,s'} \pi(a_1|s) P_M(s'|s,\boldsymbol{a}) \left(\pi_B(a_2|s) - \mu(a_2|s)\right) \left(r(s,\boldsymbol{a},s') + \gamma V^{\pi,\mu}(s')\right) \\ &\quad + \sum_{a_1,a_2,s'} \pi(a_1|s)\pi_B(a_2|s) P_M(s'|s,\boldsymbol{a}) \gamma \varepsilon(s').\end{aligned} \tag{11}$$

□

Proof of Theorem 4.3

Theorem 4.3 We use $\varepsilon = \sum_{s_0} p_0(s_0)\varepsilon(s_0)$ to denote the overall performance gap between policy profile (π, μ) and (π, π_B). In any 2-player fully observable game, for all reward functions, $\varepsilon = 0$ if and only if $\pi_B(a|s) = \mu(a|s), \forall s$, s.t. $d_{\pi,\pi_B}(s) > 0$.

Proof. Sufficiency. Whenever $\pi_B(a|s) = \mu(a|s)$ holds, the first term in equation 1 is 0. If $\forall s$ s.t. $d_{\pi,\pi_B}(s) > 0$, $\pi_B(a|s) = \mu(a|s)$, according to Proposition 4.2, by expanding the expression for $\varepsilon(s)$ recursively, we have for $\forall s$ s.t. $d_{\pi,\pi_B}(s) > 0$, $\varepsilon(s) = 0$. Therefore, $\varepsilon = \sum_{s_0} p(s_0)|\varepsilon(s_0)| = 0$.

Necessity. We first show by contradiction that $\varepsilon = \sum_{s_0} p(s_0)\varepsilon(s_0)$ requires $\forall s$ s.t. $p_0(s) > 0$, $\pi_B(a|s) = \mu(a|s)$. If $\pi_B(a|s) \neq \mu(a|s)$, according to Proposition 4.2, we can change the reward function $r(s, \boldsymbol{a}, s')$ to change the value of $\varepsilon(s)$, because the reward function is arbitrary. So $\varepsilon = \sum_{s_0} p(s_0)\varepsilon(s_0)$ does not hold anymore, and this is a contradiction. Therefore, $\varepsilon(s) = \sum_{s_0,s_1} p(s_0)\pi(a_1|s_0)\pi_B(a_2|s_0)P_M(s_1|s, \boldsymbol{a})\varepsilon(s_1) = \mathbb{E}_{s_1\sim(\pi,\pi_B)}[\varepsilon(s_1)]$. Using the same arguments, it can be shown that $\forall s,$ s.t. $Pr(s_1 = s) > 0$, $\pi_B(a|s) = \mu(a|s)$. By expanding recursively, the statement is proved. □

Proof of Theorem 4.7

We first prove a Lemma.

Lemma A.1. *In a two-player game, suppose that π is player 1's policy, α and μ are player 2's policies. We use $\pi^{joint}_{(\pi,\alpha)}$ and $\pi^{joint}_{(\pi,\mu)}$ to denote the joint policy profiles (π, α) and (π, μ). Then we have $D_{\mathrm{KL}}\left(\pi^{joint}_{(\pi,\alpha)}(\cdot|s)\|\pi^{joint}_{(\pi,\mu)}(\cdot|s)\right) = D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right)$, where D_{KL} denotes KL divergence.*

Proof. According to definition,

$$\begin{aligned} D_{\mathrm{KL}}\left(\pi^{\text{joint}}_{(\pi,\alpha)}(\cdot|s)\|\pi^{\text{joint}}_{(\pi,\mu)}(\cdot|s)\right) &= \sum_{a_1,a_2} \pi(a_1|s)\alpha(a_2|s)\log\frac{\pi(a_1|s)\alpha(a_2|s)}{\pi(a_1|s)\mu(a_2|s)} \\ &= \sum_{a_1}\pi(a_1|s)\sum_{a_2}\alpha(a_2|s)\log\frac{\alpha(a_2|s)}{\mu(a_2|s)} \\ &= \sum_{a_1}\pi(a_1|s)D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right) \\ &= D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right). \end{aligned} \tag{12}$$

□

Additionally, we use the Theorem 1 in [36] and we provide a restatement. The original paper [36] minimizes accumulated discounted cost. In contrast, an agent maximizes its return in our paper.

Theorem A.2. ***(Theorem 1 in [36])*** *Let $\epsilon = \max_s |\,\mathbb{E}_{a\sim\pi_2(a|s)}[A^{\pi_1}(s,a)]|$, then*

$$J(\pi_2) - J(\pi_1) \geq \mathbb{E}_{s\sim d_{\pi_1}(s)}\,\mathbb{E}_{a\sim\pi_2(a|s)}\left[A^{\pi_1}(s,a)\right] - \frac{2\epsilon\gamma}{(1-\gamma)^2}\max_s D_{\mathrm{KL}}\left(\pi_1(\cdot|s)\|\pi_2(\cdot|s)\right). \tag{13}$$

Finally, we prove Theorem 4.7.

Theorem 4.7 In a two-player game, suppose that π is player 1's policy, α and μ are player 2's policies. Assume that $\max_s D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right) \leq \delta$. Let R_M be the maximum magnitude of return obtainable for player 1 in this game, and let γ be the discount factor. We use $J(\pi, \mu)$ and $J(\pi, \alpha)$ to denote the return for player 1 when playing policy profiles (π, μ) and (π, μ) respectively. Then,

$$J(\pi,\mu) - J(\pi,\alpha) \geq -R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right). \tag{14}$$

Proof. According to Theorem A.2, we have

$$\begin{aligned} J(\pi,\mu) - J(\pi,\alpha) \geq\ & \mathbb{E}_{s\sim d_{(\pi,\alpha)}}\,\mathbb{E}_{a_1\sim\pi,a_2\sim\mu}[A^{(\pi,\alpha)}(s,\boldsymbol{a})] \\ & - \frac{2\epsilon\gamma}{(1-\gamma)^2}\max_s D_{\mathrm{KL}}\left(\pi^{\text{joint}}_{(\pi,\alpha)}(\cdot|s)\|\pi^{\text{joint}}_{(\pi,\mu)}(\cdot|s)\right), \end{aligned} \tag{15}$$

where $\epsilon = \max_s |\mathbb{E}_{a_1\sim\pi, a_2\sim\mu}[A^{(\pi,\alpha)}(s,\boldsymbol{a})]| \geq 0$. Use Lemma A.1, we have

$$\begin{aligned}
J(\pi,\mu) - J(\pi,\alpha) &\geq \mathbb{E}_{s\sim d_{(\pi,\alpha)}} \mathbb{E}_{a_1\sim\pi, a_2\sim\mu}[A^{(\pi,\alpha)}(s,\boldsymbol{a})] - \frac{2\epsilon\gamma}{(1-\gamma)^2} \max_s D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right) \\
&\geq -\epsilon - \frac{2\epsilon\gamma}{(1-\gamma)^2} \max_s D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right) \\
&= -\epsilon\left(1 + \frac{2\gamma}{(1-\gamma)^2} \max_s D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right)\right) \\
&\geq -\epsilon\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right).
\end{aligned} \tag{16}$$

Next, we use KL divergence to bound ϵ. According to definition, $\left|Q^{(\pi,\alpha)}(s,\boldsymbol{a})\right| \leq R_M$. So we have,

$$\begin{aligned}
\epsilon &= \max_s |\mathbb{E}_{a_1\sim\pi, a_2\sim\mu}[A^{(\pi,\alpha)}(s,\boldsymbol{a})]| \\
&= \max_s \Big| \sum_{a_1} \pi(a_1|s) \sum_{a_2} \mu(a_2|s) \left(Q^{(\pi,\alpha)}(s,\boldsymbol{a}) - V^{(\pi,\alpha)}(s)\right) \Big| \\
&= \max_s \Big| \sum_{a_1} \pi(a_1|s) \sum_{a_2} \left(\mu(a_2|s) - \alpha(a_2|s)\right) Q^{(\pi,\alpha)}(s,\boldsymbol{a}) \Big| \\
&\leq \max_s \sum_{a_1} \pi(a_1|s) \sum_{a_2} \big|\mu(a_2|s) - \alpha(a_2|s)\big| \big|Q^{(\pi,\alpha)}(s,\boldsymbol{a})\big| \\
&\leq \max_s \sum_{a_1} \pi(a_1|s) \sum_{a_2} \big|\mu(a_2|s) - \alpha(a_2|s)\big| R_M
\end{aligned} \tag{17}$$

With Pinsker's inequality [4], the total variation distance is bounded by KL divergence:

$$\delta(\alpha(\cdot|s), \mu(\cdot|s)) = \frac{1}{2}\|\alpha(\cdot|s) - \mu(\cdot|s)\|_1 \leq \sqrt{\frac{1}{2} D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right)}. \tag{18}$$

Therefore,

$$\begin{aligned}
\epsilon &\leq \max_s \sum_{a_1} \pi(a_1|s) \cdot 2R_M \delta(\alpha(\cdot|s), \mu(\cdot|s)) \\
&\leq \sqrt{2} R_M \cdot \max_s \sqrt{D_{\mathrm{KL}}\left(\alpha(\cdot|s)\|\mu(\cdot|s)\right)} \\
&\leq R_M \sqrt{2\delta}.
\end{aligned} \tag{19}$$

Put inequality 19 back into 16, and we get

$$J(\pi,\mu) - J(\pi,\alpha) \geq -R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right). \tag{20}$$

□

Proof of Theorem 4.8

Theorem 4.8 Let π^* be the optimal risk-free offline adaptation policy at the convergence of the optimization of objective 2, and let $\tilde{\pi}$ be the policy at the convergence of objective 5. Then the worst-case adaptation performance of $\tilde{\pi}$ is near-optimal:

$$\min_{\mu\in\mathcal{C}_D} J(\pi^*,\mu) \geq \min_{\mu\in\mathcal{C}_D} J(\tilde{\pi},\mu) \geq \min_{\mu\in\mathcal{C}_D} J(\pi^*,\mu) - R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right). \tag{21}$$

Proof. According to definition, π^* is the solution to the optimization objective 2, so $\min_{\mu\in\mathcal{C}_D} J(\pi^*,\mu) \geq \min_{\mu\in\mathcal{C}_D} J(\tilde{\pi},\mu)$.

Suppose that (π^*,μ^*) and $(\tilde{\pi},\tilde{\alpha})$ are the solutions to objectives 2 and 5 respectively. For $\forall\mu \in \mathcal{C}_D$, let $\mathcal{F}(\mu) = \{\alpha|\forall s \notin D, \alpha(\cdot|s) = \mu(\cdot|s); \max_{s\in D} D_{\mathrm{KL}}(\pi_B(\cdot|s)\|\alpha(\cdot|s)) \leq \delta\}$ be the set of corresponding α policies which are identical to μ on OOD states. Observe that $\max_s D_{\mathrm{KL}}(\pi_B(\cdot|s)\|\alpha(\cdot|s)) \leq \delta$ also holds. Therefore, according to Theorem 4.7, $\forall\pi, \forall\mu \in$

$\mathcal{C}_D, \forall \alpha \in \mathcal{F}(\mu)$, $J(\pi, \alpha) \geq J(\pi, \mu) - R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right)$. Taking the minimization over μ and α, we get $\forall \pi, \min_\mu \min_{\alpha \in \mathcal{F}(\mu)} J(\pi, \alpha) \geq \min_\mu J(\pi, \mu) - R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right)$. Observe that the left part is equivalent to $\min_{\alpha:\max_{s\in D} D_{\mathrm{KL}}(\alpha(\cdot|s)\|\pi_B(\cdot|s))\leq\delta} J(\pi, \alpha)$. Taking the maximization over π for both sides, we get $J(\tilde{\pi}, \tilde{\alpha}) \geq J(\pi^*, \mu^*) - R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right)$.

Let $J(\tilde{\pi}, \mu') = \min_{\mu\in\mathcal{C}_D} J(\tilde{\pi}, \mu)$. Observe that μ' also satisfies the KL divergence constraint: $\max_{s\in D} D_{\mathrm{KL}}(\pi_B(\cdot|s)\|\mu'(\cdot|s)) \leq \delta$, and $(\tilde{\pi}, \tilde{\alpha})$ is the optimal solution to objective 5, so $J(\tilde{\pi}, \mu') \geq J(\tilde{\pi}, \tilde{\alpha})$. So we get $\min_{\mu\in\mathcal{C}_D} J(\tilde{\pi}, \mu) \geq J(\tilde{\pi}, \tilde{\alpha}) \geq J(\pi^*, \mu^*) - R_M\sqrt{2\delta}\left(1 + \frac{2\gamma\delta}{(1-\gamma)^2}\right)$. □

B Related Work

RL in multi-agent games. Multi-agent reinforcement learning (MARL) under the centralized training decentralized execution (CTDE) paradigm have shown promising performances in learning coordination behavior for cooperative multi-agent tasks such as SMAC [35] and Google Football [21]. Representative works consist of policy gradient algorithms [6, 48], and value decomposition methods [34, 40]. There also have been extensive research in competitive environments such as Texas hold'em and MOBA games. Some works design theoretically sounded RL algorithms which are guaranteed to converge to approximate NE [3]. Some works propose RL methods based on self-play [45, 46, 50] to train max-min policies which show strong empirical performances.

Offline RL. Offline reinforcement learning [22] learns purely from batched data. Representative works take a conservative view of out-of-distribution state-action pairs to mitigate the distributional shift problem. BCQ [8] uses VAE to propose candidate actions that are within the support of dataset. IQL [19] uses expectile regression to learn Q function to avoid querying out-of-distribution actions. CQL [20] puts regularization on the learning of Q function to penalize out-of-distribution actions.

Opponent exploitation. There have been extensive study on various aspects of opponent exploitation [1, 28]. Some employ deep reinforcement learning to exploit the opponent in continuous control tasks using policy gradients estimated from direct interaction [2, 10, 12]. Some works, including RNR [17] and SES [24] study safe exploitation given an estimated opponent model. They keep the exploitation policy close to NE in order to minimize the loss if the opponent changes its policy adversarially. Some work, for example, GSCU [7], studies zero-shot generalization ability to exploit totally unknown opponents. To our best knowledge, there lacks a thorough investigation into the offline adaptation problem.

Robust RL. Formulated as robust-MDP (RMDP), robust RL [32, 30] is dedicated to learn a robust policy against perturbation in environment dynamics, e.g., observation noise, action delay. Formally, the unknown transition model in testing environment lies in an uncertainty set $\mathcal{P}$ which contains all models within a certain distance from the training model. Robust RL solves a max-min problem, optimizing the worst-case performance against any transition model in $\mathcal{P}$. Some works uses adversarial training [32, 16]. RORL [42] uses smoothing on the Q function to learn robust policy. The similarity to our work is that, our offline adaptation problem also optimizes the worst-case performance against the target's behavior on out-of-distribution states. However, we do not have any distance constraints on the target's policy on out-of-distribution states. To our best knowledge, our paper is the first to investigate into the offline adaptation problem.

C Algorigthm

The CSP algorithm is illustrated in Algorithm 1. The proxy model is trained adversarially against our agent, therefore, we set the proxy's reward function to be the negative of our agent's reward. In our experiments on an n-player environment, we assume that we control n_1 players, while the target controls n_2 players, and $n_1 + n_2 = n$. Therefore, we use MAPPO [48] as the basic learning algorithm for players of both sides, since players from the same side are fully cooperative. MAPPO deals with the cooperative multi-agent reinforcement learning problem through learning a centralized value function conditioned on global state, and a decentralized control policy conditioned on local observations. We use self-play with alternative update to learn both adaptation policy π, and target's

proxy model μ simultaneously. We use a soft behavior cloning regularization term to minimize the KL-divergence between proxy model μ and target policy π_B.

Algorithm 1 Constrained Self-Play (CSP)

Input: dataset D, environment env, learning rate α, regularization coefficient C_{BC}
Output: adaptation policy π, target's policy proxy μ

Initialize adaptation policy π, critic v^π; and target's policy proxy μ, critic v^μ
while not converged **do**
  Collect trajectories $\{\mathcal{T}_i\} \leftarrow \text{rollout}(\pi, \mu; env)$
  if our turn **then**
    For each trajectory $\mathcal{T}_i$, calculate return target R_t^γ and advantage $\hat{A}_t$ of ***our side*** using GAE with value function v^π for each step t
    $v^\pi \leftarrow v^\pi - \alpha \nabla_{v^\pi} \frac{1}{|\cup \mathcal{T}_i|} \sum_{(s_t, R_t^\gamma) \sim \cup \mathcal{T}_i} (R_t^\gamma - v^\pi(s_t))$
    $\pi \leftarrow \pi + \alpha \nabla_\pi \frac{1}{|\cup \mathcal{T}_i|} \sum_{(o_t, a_t, \pi_{old}, \hat{A}_t) \sim \cup \mathcal{T}_i} \min \left(\frac{\pi(a_t|o_t)}{\pi_{old}(a_t|o_t)} \hat{A}_t, \right.$
      $\left. \text{clip} \left(\frac{\pi(a_t|o_t)}{\pi_{old}(a_t|o_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right)$
  else
    For each trajectory $\mathcal{T}_i$, calculate return target R_t^γ and advantage $\hat{A}_t$ of ***target agent side*** using GAE with value function v^μ for each step t
    $\mathcal{B} \leftarrow$ sample a random batch from D
    $v^\mu \leftarrow v^\mu - \alpha \nabla_{v^\mu} \frac{1}{|\cup \mathcal{T}_i|} \sum_{(s_t, R_t^\gamma) \sim \cup \mathcal{T}_i} (R_t^\gamma - v^\mu(s_t))$
    $\mu \leftarrow \mu + \alpha \nabla_\mu \left\{ \frac{1}{|\cup \mathcal{T}_i|} \sum_{(o_t, a_t, \pi_{old}, \hat{A}_t) \sim \cup \mathcal{T}_i} \min \left(\frac{\mu(a_t|o_t)}{\mu_{old}(a_t|o_t)} \hat{A}_t, \right. \right.$
      $\left. \left. \text{clip} \left(\frac{\mu(a_t|o_t)}{\mu_{old}(a_t|o_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) - C_{\text{BC}} \cdot \frac{1}{|\mathcal{B}|} \sum_{(o,a) \in \mathcal{B}} - \log \mu(a|o) \right\}$
  end if
end while

D Experiment on Didactic Maze

We show experiment details of the Maze example in this section. The learning curve in case 1 is shown in Figure 5. In the BC-First algorithm, since the dataset only contains trajectories that P1 $\rightarrow$ 64, the BC model makes a wrong and risky generalization that assumes P1 always goes downwards to 64 no matter how P2 acts. Therefore, the P2 policy learns to exploit this "weakness" by playing P2 $\rightarrow$ 16, and achieves 16 in training phase. However, during testing, P2 obtains -4 reward through trying to exploit an nonexistent weakness of P1.

Our algorithm avoids the trap of imaginary weakness, and learns to safely win the game, while the BC-First method makes risky exploitations which will not work in evaluation finally. Our algorithm keeps conservative for states outside dataset, and admits the possibility that P1 could go leftwards to win the game if P2 does not plays P2 $\rightarrow$ 1.

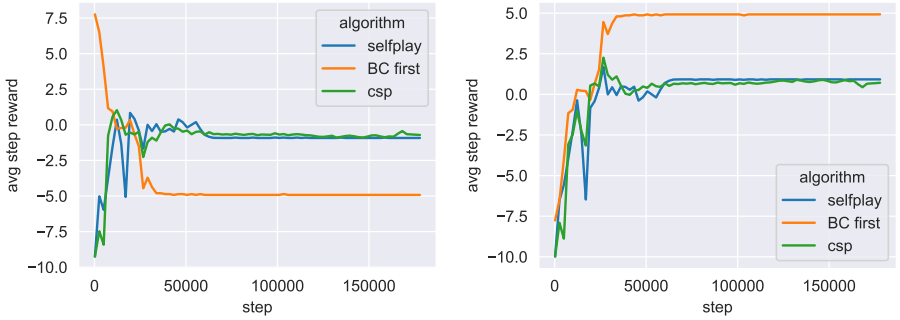

Figure 5: The training curve of amortized average per step reward in maze game case 1 in **training** (against the proxy). The x-axis represents training steps. **Left.** Average per step reward for P1. **Right.** Average per step reward for P2.

The learning curve in case 2 is shown in Figure 6. As can be seen in Figure 6, our algorithm quickly learns to exploit while the max-min strategy produced by Self-Play fails to.

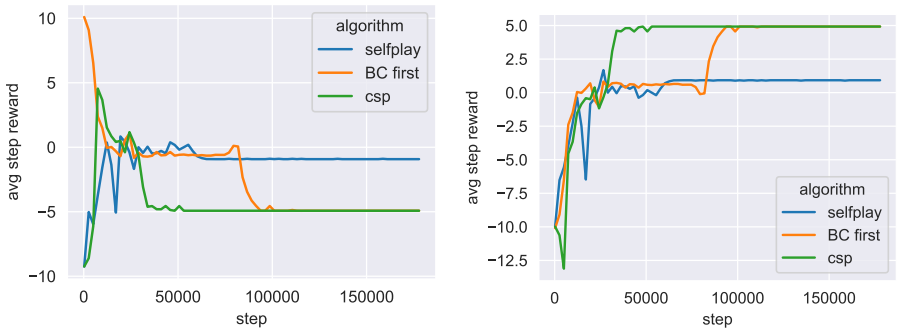

Figure 6: The training curve of amortized average per step reward in maze game case 2 in **training** (against the proxy). The x-axis represents training steps. **Left.** Average per step reward for P1. **Right.** Average per step reward for P2.

We use the same set of hyper-parameters (including the same coefficient C_{BC} for the BC regularization term) for these two cases. The hyper-parameters are shown in Table 5. The dataset $D_1 = \{\text{trajectories}(P1 \to 64, P2 \to 256)\}$ in case 1, and our algorithm learns to play P2 $\to$ 1; while in case 2, given dataset $D_2 = \{\text{trajectories}(P1 \to 64, P2 \to 256), \text{trajectories}(P1 \to 64, P2 \to 16)\}$, our algorithm learns to play P2 $\to$ 16. We use the same set of hyper-parameters for both cases. Therefore, our algorithm produces different policies simply because of different datasets given. It further validates that our algorithm is able to extract useful information from dataset, and only perform risk-free adaptations.

Table 5: Hyper-parameters for maze.

ppo_epoch	1	num_mini_batch	1	entropy_coef	0.3
use_gae	True	gamma	0.99999	gae_lambda	0.95
critic_lr	7e-4	lr	7e-4	weight_decay	0
adam_eps	1e-5	n_rollout_threads	20	ppo_episode_length	12
data_chunk_length	12	steps	1.8K	max_grad_norm	0.5
bc_regularization_coef	10	bc_batch_size	8	network	MLP

E Conservative Offline Adaptation for Cooperative Tasks

We evaluate our method in competitive games to perform opponent exploitation. Nevertheless, our method is also applicable to cooperative environments. The objective of CSP in cooperative games maintains the same min-max structure because it promotes adherence to the policy exposed by dataset for states within the dataset, while aiming to optimize for worst-case performance for states not in the dataset. Therefore, in cooperative games, the policy regularization term incentivizes the adaptation agent to collaborate with the target agent on states within the dataset, and the minimization over opponent proxy μ encourages the adaptation agent to be conservative and not to rely on the target agent's OOD policy. Note that in objective 5 and Algorithm 1, no matter whether the game is competitive or cooperative, the minimization over μ in training can be achieved by setting the target agent's reward function to be the negative of our adaptation agent's reward function. We then train μ to maximize this reward with policy regularization in self-play.

Although our method is applicable to cooperative environments as well, the challenges caused by offline policy adaptation are more significant in competitive games. Recall that conservative offline adaptation optimizes

$$\max_{\pi} \min_{\mu} J(\pi, \mu), \ s.t. \ \mu \in \mathcal{C}_D. \tag{22}$$

For cooperative tasks, it requires that the teammate μ will not cooperate on states outside dataset in cooperative games. Therefore, for the adaptation policy, staying within dataset could be a trivial and

near-optimal solution for most cooperative problems if we would like to maximize the worst-case performance against any dataset-consistent teammate.

F Potential Negative Social Impacts

Inappropriate use of exploitation algorithms may result in negative social impacts. An example of its negative impact is the exacerbation of inequality in society. Companies who have access to large amounts of data can utilize their customers more effectively. However, our algorithm is general-purpose and depends on pre-collected data. We advocate for strict laws and regulations that protect user privacy data to avoid negative impacts. It is recommended that products utilizing exploitation algorithms are made public and supervised.

G Experiment Details

G.1 Environments

The experiment environments are illustrated in figure 7.

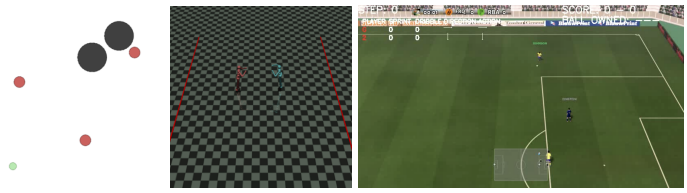

Figure 7: Illustration of experiment environments used in this paper. From left to right: (1) predator prey in MPE, (2) YouShallNotPassHumans in MuJoCo, (3) Google Football.

Predator-Prey There are one good agent, three adversarial agents and two obstacles in the predator-prey environment. The good agent is faster and receive a negative reward for being hit by adversaries while adversaries are slower and are rewarded for hitting the good agent. All agents are initialized randomly in the environment.

YouShallNotPassHumans The YouShallNotPassHumanoids environment [2, 10] creates a two-player competitive game based on MuJoCo, where one humanoid (the runner) is aimed at passing the other humanoid (the blocker) to reach the red line on the opposite side. The environment itself is challenging because it has high dimensional observation space and requires continuous control. In our experiments, we assume that our agent acts as the runner, while the blocker is the target opponent to exploit. We use four independently pre-trained blocker models as opponent targets. For fairness, we generate these targets with exactly four random seeds without any selection. Since the exploitability of different opponent models can vary significantly, results with respect to different opponents are not directly comparable. So we report results with respect to all targets.

Google Football Google Football [21] is a popular and challenging benchmark for MARL. In our experiments, our adaptation policy controls all players in one team, while the opponent target policy controls the other team. Since all players in the same team are fully cooperative, we use MAPPO [48] to learn decentralized policy. We conduct experiments in 4 scenarios: *academy_3_vs_1_with_keeper* (3vs1, where our agent acts as either defender or attacker), *academy_run_pass_and_shoot_with_keeper* (RPS, defender), and *academy_counterattack_easy* (defender). We report the winning rates of adaptation policies for 5 independently pre-trained opponent targets. For attacker, winning rate refers to the percentage of episodes that the attacker scores, while for defender, it refers to the percentage of episodes that the defender prevents the attacker from scoring.

G.2 Hyper-parameters

In our experiments, we use MAPPO [48] as the base RL algorithm to learn policies on both sides (ours and the target's). MAPPO is an extension of single agent PPO to multi-agent reinforcement learning

within the centralized training decentralized evaluation (CTDE) paradigm. It learns a centralized value function conditioning on global state to promote coordination among the agents. We selected MAPPO as the base learning algorithm due to its simplicity and empirical strong performance.

For the BC-First method, we use exactly the same MAPPO (with the same set of hyper-parameters and the same number of training samples) as CSP to train our adaptation policy, while keeping the target's proxy fixed. In order to ensure that the the performances of BC-First method are not encumbered by an under-trained proxy model, we use the same network structure for the proxy model as the real target model, and train enough steps to make sure that behavior cloning has converged. For the Google Football environment, we use the same hyper-parameters as reported in the MAPPO paper [48] for all scenarios. The hyper-parameters are listed in Table 6, 7, and 8.

Table 6: Hyper-parameters for predator-prey in MPE.

ppo_epoch	10	num_mini_batch	1	entropy_coef	0.01
use_gae	True	gamma	0.99	gae_lambda	0.95
critic_lr	7e-4	lr	7e-4	weight_decay	0
adam_eps	1e-5	n_rollout_threads	128	ppo_episode_length	50
data_chunk_length	10	steps	5M	max_grad_norm	0.5
bc_regularization_coef	0.003	bc_batch_size	8	network	RNN

Table 7: Hyper-parameters for MuJoCo.

ppo_epoch	4	num_mini_batch	1	entropy_coef	0.01
use_gae	True	gamma	0.99	gae_lambda	0.95
critic_lr	7e-4	lr	7e-4	weight_decay	0
adam_eps	1e-5	n_rollout_threads	100	ppo_episode_length	200
data_chunk_length	10	steps	40M	max_grad_norm	0.5
bc_regularization_coef	0.1	bc_batch_size	256	network	MLP

Table 8: Hyper-parameters for Google Football.

ppo_epoch	15	num_mini_batch	2	entropy_coef	0.01
use_gae	True	gamma	0.99	gae_lambda	0.95
critic_lr	7e-4	lr	7e-4	weight_decay	0
adam_eps	1e-5	n_rollout_threads	50	ppo_episode_length	200
data_chunk_length	10	steps	25M	max_grad_norm	0.5
bc_regularization_coef	5.0	bc_batch_size	256	network	RNN

G.3 Computing Resources

Each seed is run on a GPU server with one NVIDIA P100 GPU, and Intel(R) Xeon(R) Gold 6145 CPU @ 2.00GHz CPU. Each run can finish within 24 hours.

G.4 Target Models & Dataset Collection

For all the environments, we user different runs of self-play to get the group of targets. In order to ensure a fair comparison, we use the exactly the same number of random seeds to generate these targets, **without any selection**. In our experiments, we observe that different runs with different seeds can produce diverse targets. For instance, in Google Football, different targets may have different tendencies to pick which side to start a attack (left or right). Moreover, as can be seen from Table 4, the difficulties to exploit these targets are diverse. For MuJoCo and Google Football, we use dataset consisting of 5 trajectories. We collect these trajectories using the target model together with a rollout policy. We observe that the 3 kinds of rollout policies: (1) random policy, (2) environment's bot, (3) target itself, do not have significant impacts on the experiment results in the MuJoCo and Google Football environments.

G.5 Visualization

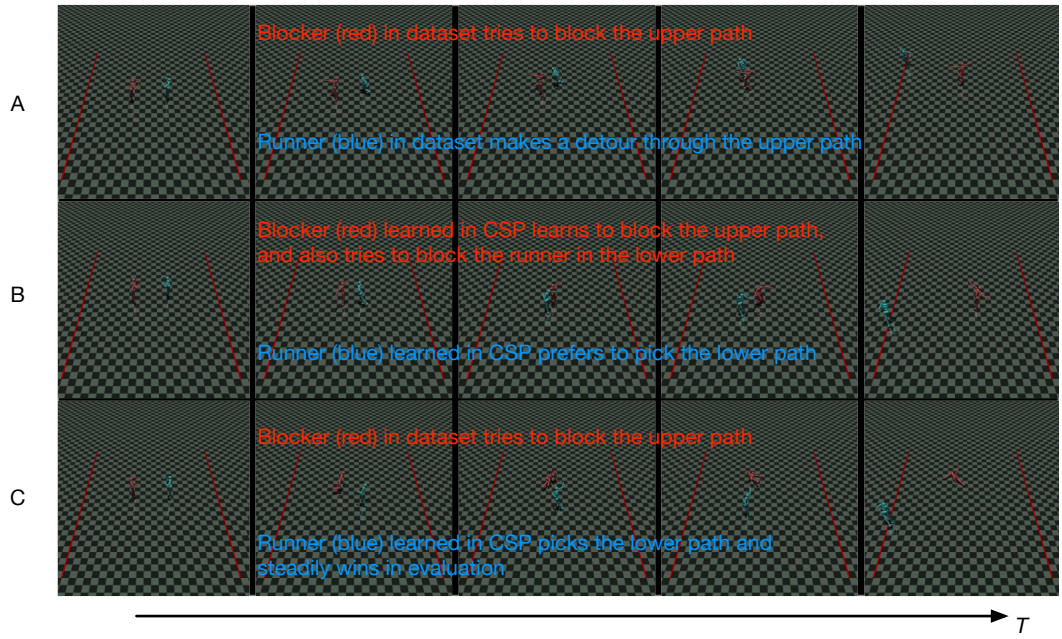

Figure 8: The visualization of policy behavior in MuJoCo. **Top.** The trajectory in dataset, where the blocker is the real opponent policy π_B and the runner is the policy which collects the dataset. **Middle.** The trajectory during the training of CSP. Both agents are controlled by CSP. **Bottom.** The evaluation of CSP's runner with the real opponent blocker.

In Figure 8, we further consolidate our claim by visualizing the policy behavior in MuJoCo. As can be seen in this example, in a trajectory contained in dataset, the runner (blue agent) typically makes a detour through the upper path to avoid being pushed down by the blocker (red agent). The opponent blocker also has the tendency to walk upwards to stop the runner. The runner trained by CSP learns to exploit the policy represented by the dataset, and prefers making a detour through the lower path, as can be seen from the middle and bottom lines of replays. Although the behavior of runner going downwards is not contained in the dataset, the blocker trained by CSP can still learns to stop such runner, which makes the runner even more robust and stronger than the real target model. Therefore, when confronted with the real opponent blocker, who only knows how to defend the upper path, the runner trained by CSP steadily wins the game.

G.6 Reward.

For predator-prey, we use the environment's original reward. For MuJoCo, we use dense rewards for locomotion learning as in previous works [2, 10]. In Google Football, we use both the sparse scoring reward as well as the dense checkpoint reward which awards the attacker according to the ball handler's distance to the goal. Although the rewards for the attacker are provided by the environment, Google Football does not provide rewards for the defender team in academy scenarios. In our experiments, we just use the negative of the attacker's reward to train defender's policy to make the game zero-sum.

G.7 Explanation of Training & Testing Performances

The observed lower training performance of CSP than the baseline, as shown in Figure 1, is expected behavior since the opponent model used by CSP during training differs from that of BC-First. In BC-First, the opponent model is solely pre-trained on a dataset and remains fixed, leading to significant vulnerabilities in out-of-distribution states which can be easily exploited. Conversely, in CSP, the opponent model is trained in an adversarial manner simultaneously with the exploitation policy. Due to its evolutionary nature, it can compensate for its vulnerabilities and becomes significantly more

challenging to exploit. Consequently, the training performance of BC-First is higher. During testing, both exploitation policies undergo evaluation using the same real target model.