A APPENDIX

A.1 TRAINING DETAILS

Our implementation is based on the open sourced OBL code with two main modifications. We first replace the recurrent Q-learning backbone with PPO as it runs faster and requires significantly less memory. Then, we implement a synchronous method that trains all OBL levels simultaneously.

Otherwise, we use a distributed training set up similar to (Hu et al., 2021b), which we detail in the appendix. All bots are trained for 3000 epochs, each epoch consists of 1000 gradient steps. We select the checkpoint with the highest SP score.

The original OBL trains multiple levels of policies sequentially, using the output policy of the previous level as the input policy of the new level. In ADVERSITY, we train all levels simultaneously for faster wall-clock time. These policies are denoted as $\pi_0, \pi_1 \dots \pi_L$ and their corresponding belief models are denoted as $\hat{\mathcal{B}}_0, \dots, \hat{\mathcal{B}}_L$. To warm up the belief model and avoid having too many invalid samples, we first train a belief model $\hat{\mathcal{B}}_0$ on the uniform random base policy π_0 and initialize all $\hat{\mathcal{B}}_l = \hat{\mathcal{B}}_0$. Then, L policy training tasks and L belief training tasks start at the same time. The belief task of $\hat{\mathcal{B}}_l$ gets a latest copy of π_l every 50 gradient steps and the policy task of π_l gets a latest copy of $\hat{\mathcal{B}}_{l-1}$ every 50 gradient steps. The details of each individual belief follows the exact configurations of the original OBL paper and each policy task uses the PPO-OBL method described above.

For each adversary, we train a hierarchy of 7 levels, setting $\lambda = 0.25$ for $l = 1$ and decreasing by 0.08 every level (min. 0). Levels $l \leq 4$ are trained simultaneously, followed by levels $l \geq 5$, also trained simultaneously and with beliefs initialized at $\hat{\mathcal{B}}_l = \hat{\mathcal{B}}_4$. This split was forced by limitations on the concurrent compute available to the authors, but we anticipate no change in performance if all levels were trained simultaneously. The ADVERSITY numbers reported in Section 7 all refer to the highest level of the hierarchy.

A.2 POLICY TRAINING DETAILS

We use a large scale distributed training framework for policy training. To train a single policy, we run 6400 games in parallel, each adding to a centralized replay buffer. We achieve this by running 80 threads in parallel, with 80 games running per thread. All models are on GPUs and we dynamically batch all model calls in order to increase inference speed. This schema also allows games on the same thread to forward environment calls while certain games wait for GPU calls. As done in (Wang et al., 2016), when an environment terminates, each game grabs all necessary objects: observations, actions, and targets, pads everything to a length of 80 and adds it to a centralized replay buffer.

For every training step we apply the PPO update rule, but instead of using the real reward and advantage, we use the fictitious values. Every $m = 10$ training steps, we update the environment actors with the weights for the updated policy. As done in Cui et al. (2021) synchronously train our hierarchy of beliefs and policies, querying for and updating all dependencies every $p = 50$ training steps.

We utilize the same policy architecture as Hu et al. (2021b). We utilize their public-private LSTM architecture. The public observation is encoded by a one-layer feedforwards neural network followed by a LSTM. The private observation is encoded by a three-layer neural network. We combine these encodings via element wise multiplication.

For all OBL experiments we compute the target with $r = 1$ fictitious steps. We also sample the belief model $s = 10$ times and use the first sampled trajectory that doesn't violate card constraints to compute the fictitious targets. We then use a simulator to produce transitions from the valid trajectory. Like Hu et al., we discard the fictitious transition whenever the belief fails to produce a valid sample, which in practice happens on less than 1% of transitions.

Implementation

The policy is represented by a public-LSTM network π_θ with a value head and a policy head. A large number of parallel workers generate data by sampling from a slightly outdated policy $\pi_{\theta'}$ and write that data into a replay buffer $\mathcal{D}$. One datapoint in $\mathcal{D}$ is an entire trajectory τ^j. Although PPO

normally does not need a replay buffer, we still use one here to fully decouple inference and training for maximum speed. Its size is set to a small value of 1024 to minimize the instability caused by stale data. π_θ is trained with the Adam optimizer (Kingma & Ba, 2014) on minibatches of data uniformly sampled from the replay buffer. The value loss is $\mathbb{E}_{\tau^i \sim \mathcal{D}} \sum_t [r_t + \gamma V_\theta(\tau^i_{t+1}) - V_\theta(\tau^i_t)]^2$. The policy loss is $\mathbb{E}_{\tau^i \sim \mathcal{D}} \sum_t \min[r_t(\theta)\dot{A}_t, \text{clip}(r_t(\theta), 1 \pm \epsilon)\dot{A}_t]$ where $r_t(\theta) = \frac{\pi_\theta(a^i_t|\tau^i_t)}{\pi_{\theta'}(a^i_t|\tau^i_t)}$, $\dot{A}_t = \text{StopGradient}[r_t + \gamma v_\theta(\tau^i_{t+1}) - V_\theta(\tau^i_t)]$. We perform one gradient step per minibatch. We use 1-step bootstrapped value target instead of $\sum_t r_t$ because it converges significantly faster and it fits well in the OBL fictitious target computation. $\pi_{\theta'}$ is synced with π_θ every 10 gradient updates.

A.3 Belief Training Details

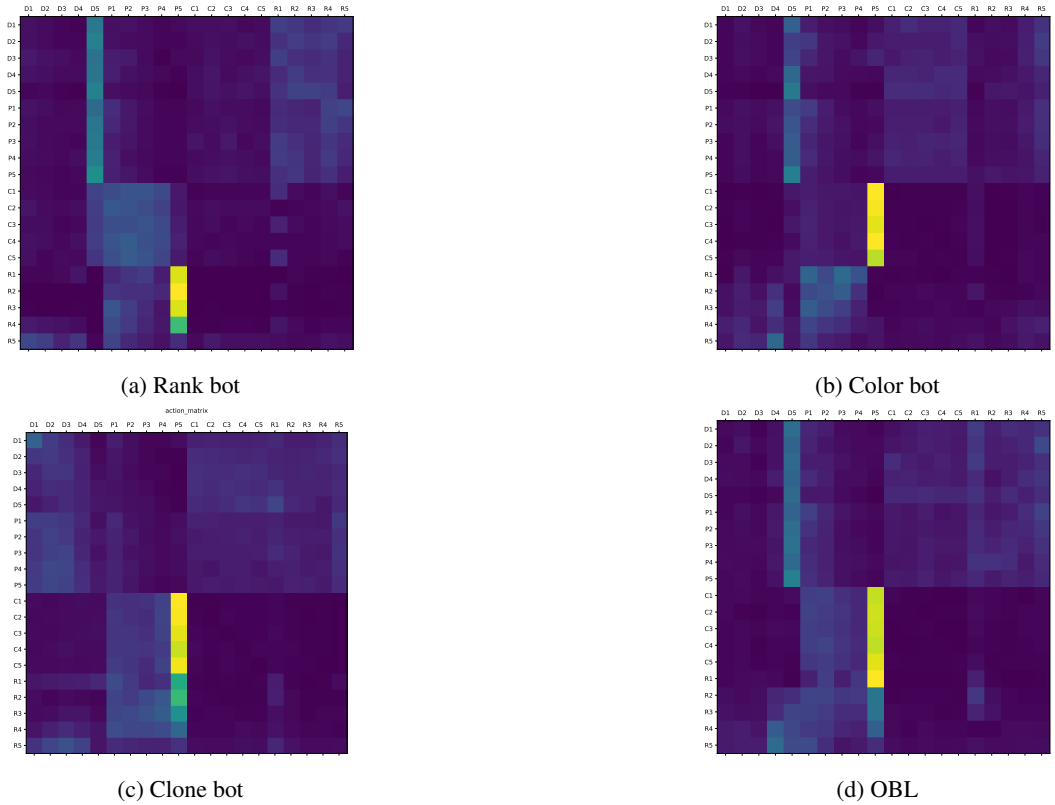
We utilize the same distributed training schema from policy training for belief training. This has also been done by (Hu et al., 2021b). As done in policy training, we query and update dependencies every $p = 50$ training steps.

For belief training we store the true hand of the player along with the observation to train the belief. For training, we train an autoregressive belief model that predicts cards oldest to newest via supervised learning. More precisely, the belief model is trained to minimize the loss

$$\mathcal{L}(\mathbf{h}|\tau^i_t) = -\sum_{k=1}^{n} \log p(h_k|\tau^i_t, h_{1:k-1}), \tag{4}$$

where h_k is the kth card in the player's hand and n is the hand size (usually 5).

A.4 Additional Results



(a) Rank bot

(b) Color bot

(c) Clone bot

(d) OBL

Figure 3: Conditional action matrices showing $p(a_{t+1}|a_t)$ for the 4 repulser policies

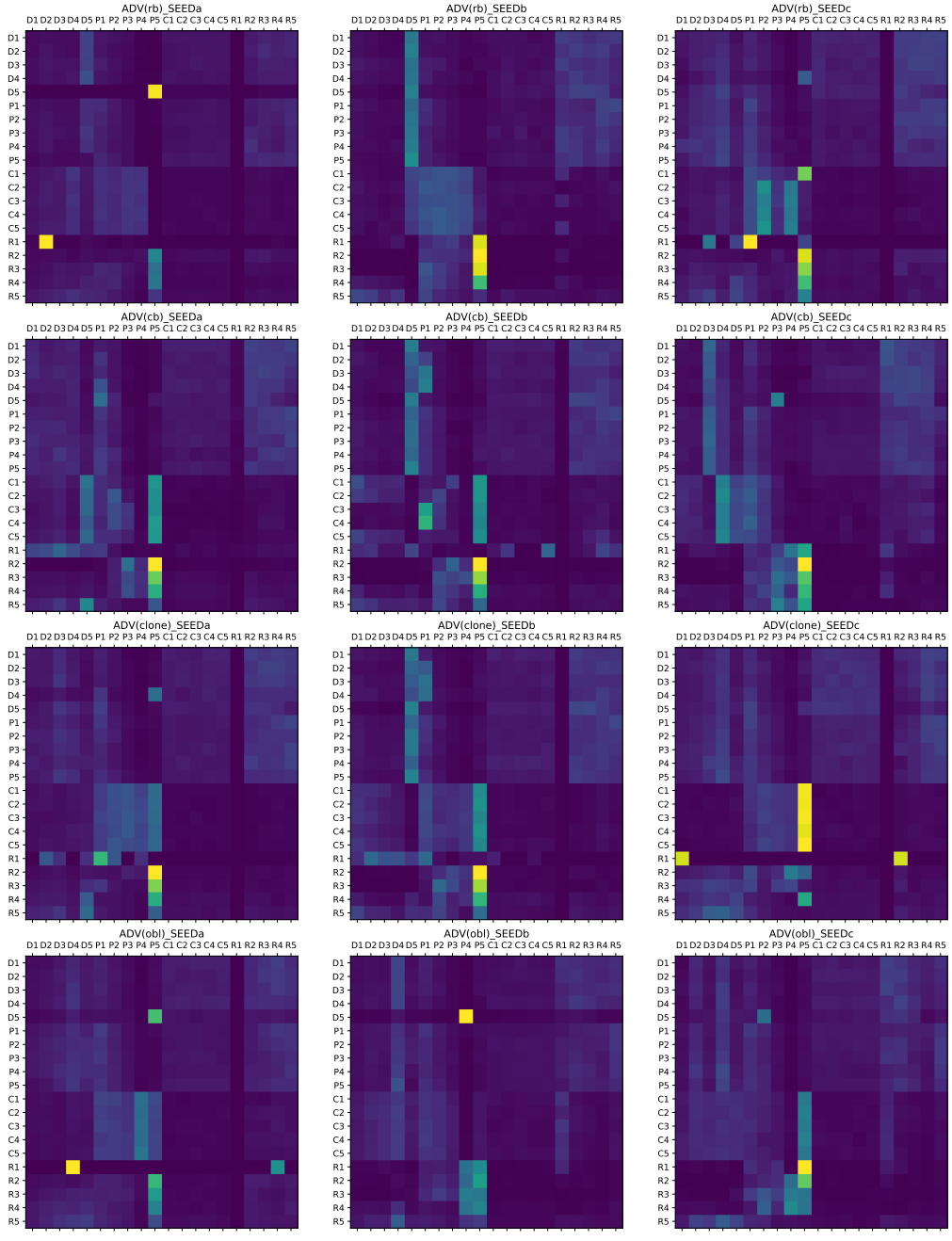


Figure 4: Action matrices for all SPWR agents.

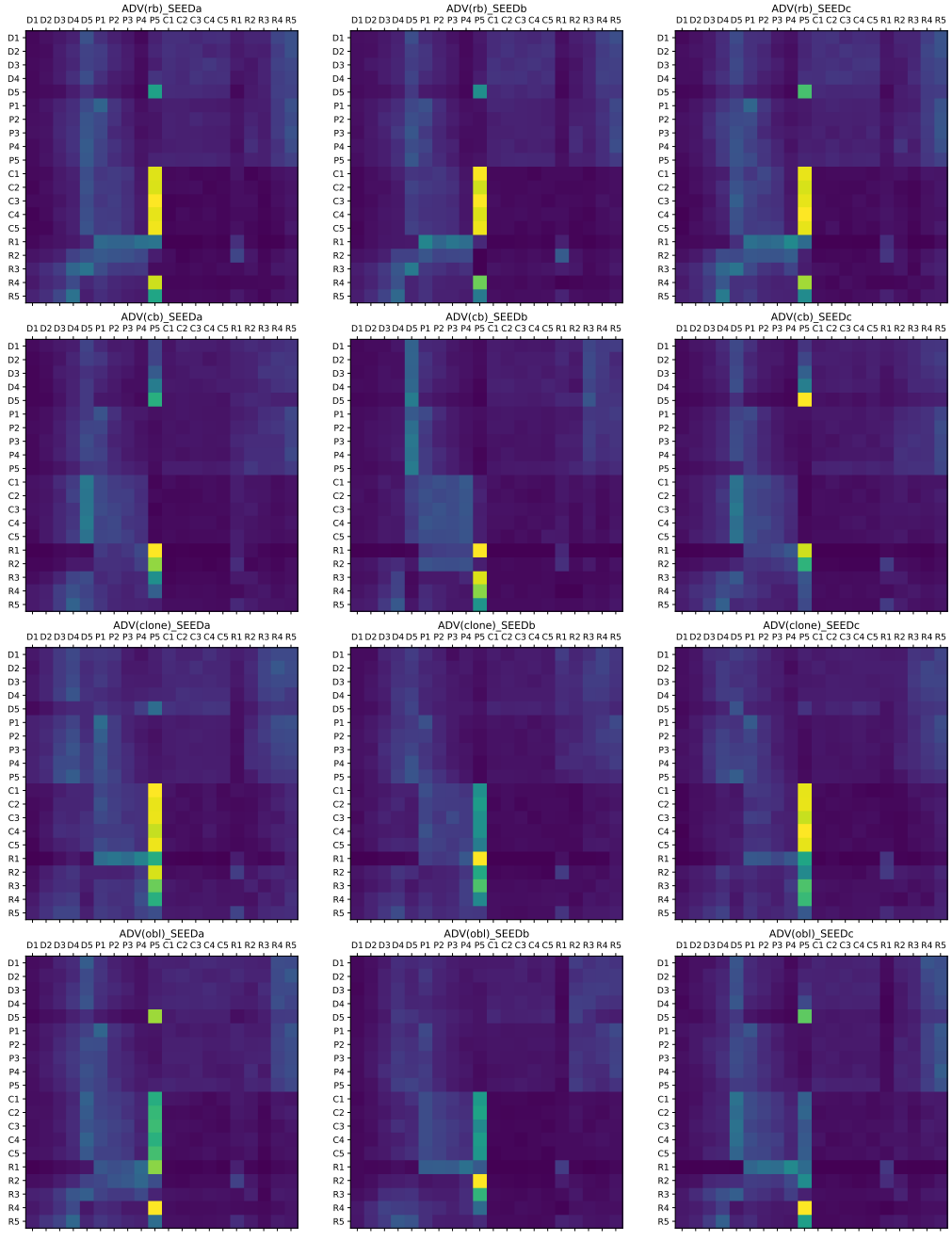


Figure 5: Action matrices for all ADVERSITY agents.

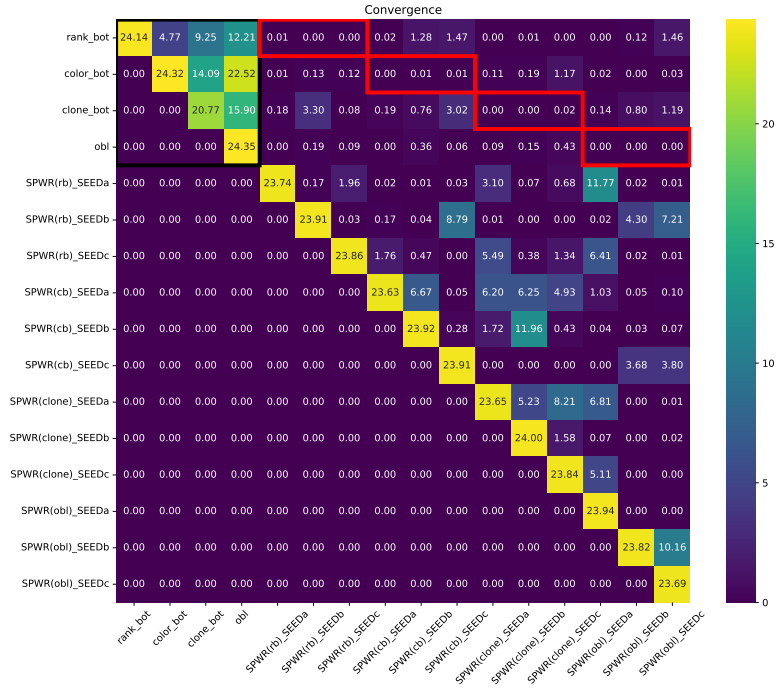


Figure 6: XP matrix of the four repulser candidates and all the SPWR bots. Red rectangles indicate pairs of the form $(X, \text{SPWR}(X))$. Numbers below the diagonal were not computed.

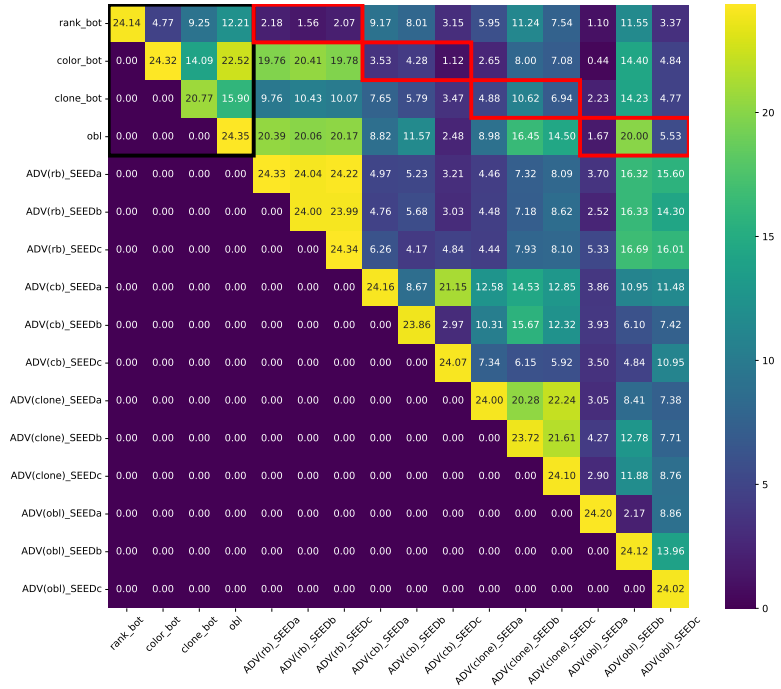


Figure 7: XP matrix of the four repulser candidates and all the ADVERSITY bots. Red rectangles indicate pairs of the form $(X, \text{Adv}(X))$. Numbers below the diagonal were not computed.