On the Utility of Learning about Humans for Human-AI Coordination

Micah Carroll
UC Berkeley
mdc@berkeley.edu

Rohin Shah
UC Berkeley
rohinmshah@berkeley.edu

Mark K. Ho
Princeton University
mho@princeton.edu

Thomas L. Griffiths
Princeton University

Sanjit A. Seshia
UC Berkeley

Pieter Abbeel
UC Berkeley

Anca Dragan
UC Berkeley

Abstract

While we would like agents that can coordinate with humans, current algorithms such as self-play and population-based training create agents that can coordinate with *themselves*. Agents that assume their partner to be optimal or similar to them can converge to coordination protocols that fail to understand and be understood by humans. To demonstrate this, we introduce a simple environment that requires challenging coordination, based on the popular game *Overcooked*, and learn a simple model that mimics human play. We evaluate the performance of agents trained via self-play and population-based training. These agents perform very well when paired with themselves, but when paired with our human model, they are significantly worse than agents designed to play with the human model. An experiment with a planning algorithm yields the same conclusion, though only when the human-aware planner is given the exact human model that it is playing with. A user study with real humans shows this pattern as well, though less strongly. Qualitatively, we find that the gains come from having the agent *adapt* to the human's gameplay. Given this result, we suggest several approaches for designing agents that learn about humans in order to better coordinate with them. Code is available at `https://github.com/HumanCompatibleAI/overcooked_ai`.

1 Introduction

An increasingly effective way to tackle two-player games is to train an agent to play with a set of other AI agents, often past versions of itself. This powerful approach has resulted in impressive performance against human experts in games like Go [33], Quake [20], Dota [29], and Starcraft [34].

Since the AI agents never encounter humans during training, when evaluated against human experts, they are undergoing a distributional shift. Why doesn't this cause the agents to fail? We hypothesize that it is because of the *competitive nature* of these games. Consider the canonical case of a two-player zero-sum game, as shown in Figure 1 (left). When humans play the minimizer role but take a branch in the search tree that is suboptimal, this only *increases* the maximizer's score.

However, not all settings are competitive. Arguably, one of the main goals of AI is to generate agents that *collaborate*, rather than compete, with humans. We would like agents that help people with the tasks they want to achieve, augmenting their capabilities [10, 6]. Looking at recent results, it is tempting to think that self-play-like methods extend nicely to collaboration: AI-human teams performed well in Dota [28] and Capture the Flag [20]. However, in these games, the advantage may come from the AI system's individual ability, rather than from *coordination* with humans. We claim that in general, collaboration is fundamentally different from competition, and will require us to go beyond self-play to explicitly account for *human* behavior.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

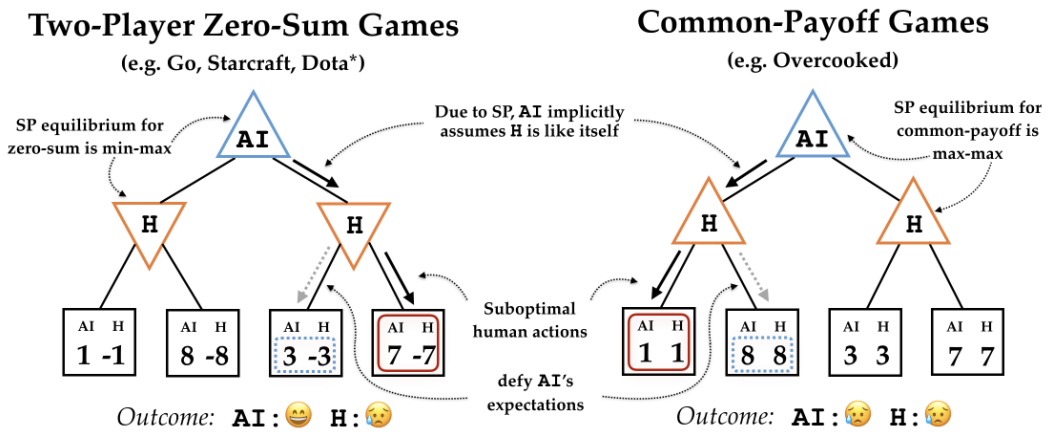


Figure 1: The impact of incorrect expectations of optimality. **Left:** In a competitive game, the agent plans for the worst case. **AI** expects that if it goes left, **H** will go left. So, it goes right where it expects to get 3 reward (since **H** would go left). When **H** suboptimally goes right, **AI** gets 7 reward: *more* than it expected. **Right:** In a collaborative game, **AI** expects **H** to coordinate with it to choose the best option, and so it goes left to obtain the 8 reward. However, when **H** suboptimally goes left, **AI** only gets 1 reward: the worst possible outcome!

Consider the canonical case of a common-payoff game, shown in Figure 1 (right): since both agents are maximizers, a mistake on the human's side is no longer an advantage, but an actual problem, *especially* if the agent did not anticipate it. Further, agents that are allowed to co-train might converge onto opaque coordination strategies. For instance, agents trained to play the collaborative game Hanabi learned to use the hint "red" or "yellow" to indicate that the newest card is playable, which no human would immediately understand [12]. When such an agent is paired with a human, it will execute the opaque policy, which may fail spectacularly when the human doesn't play their part.

We thus hypothesize that in true collaborative scenarios agents trained to play well with other AI agents will perform much more poorly when paired with humans. We further hypothesize that incorporating human data or models into the training process will lead to significant improvements.

To test this hypothesis, we introduce a simple environment based on the popular game *Overcooked* [13], which is specifically designed to be challenging for humans to coordinate in (Figure 2).We use this environment to compare agents trained with themselves to agents trained with a learned human model. For the former, we consider self-play [33], population-based training [19], and coupled planning with replanning. For the latter, we collect human-human data and train a behavior cloning human model; we then train a reinforcement learning and a planning agent to collaborate well with this model. We evaluate how well these agents collaborate with a held-out "simulated" human model (henceforth the "proxy human"), trained on a different data set, as well as in a user study.

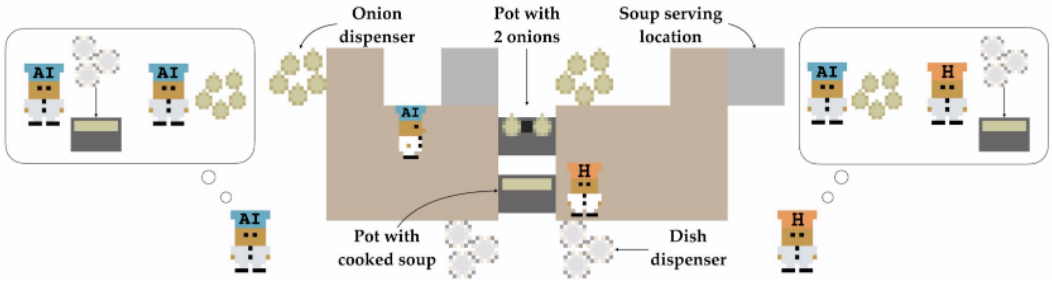


Figure 2: Our Overcooked environment. The goal is to place three onions in a pot (dark grey), take out the resulting soup on a plate (white) and deliver it (light grey), as many times as possible within the time limit. **H**, the human, is close to a dish dispenser and a cooked soup, and **AI**, the agent, is facing a pot that is not yet full. The optimal strategy is for **H** to put an onion in the partially full pot, and for **AI** to put the existing soup in a dish and deliver it. This is due to the layout structure, that makes **H** have an advantage in placing onions in pots, and **AI** have an advantage in delivering soups. However, we can guess that **H** plans to pick up a plate to deliver the soup. If **AI** nonetheless expects **H** to be optimal, it will expect **H** to turn around to get the onion, and will continue moving towards its own dish dispenser, leading to a *coordination failure*.

We find that the agents which did not leverage human data in training perform very well with themselves, and drastically worse when paired with the proxy human. This is not explained only by human suboptimality, because the agent also significantly underperforms a "gold standard" agent that has access to the proxy human. The agent trained with the behavior-cloned human model is drastically better, showing the benefit of having a relatively good human model. We found the same trends even when we paired these agents with real humans, for whom our model has much poorer predictive power but nonetheless helps the agent be a better collaborator. We also experimented with using behavior cloning directly for the agent's policy, and found that it also outperforms self-play-like methods, but still underperforms relative to methods that that leverage planning (with respect to the actual human model) or reinforcement learning with a proxy human model.

Overall, we learned a few important lessons in this work. First, our results showcase the importance of accounting for real human behavior during training: even using a behavior cloning model prone to failures of distributional shift seems better than treating the human as if they were optimal or similar to the agent. Second, leveraging planning or reinforcement learning to maximize the collaborative reward, again even when using such a simple human model, seems to already be better than vanilla imitation. These results are a cautionary tale against relying on self-play or vanilla imitation for collaboration, and advocate for methods that leverage models of human behavior, actively improve them, or even use them as part of a population to be trained with.

2 Related Work

Human-robot interaction (HRI). The field of human robot interaction has already embraced our main point that we shouldn't model the human as optimal. Much work focuses on achieving collaboration by planning and learning with (non-optimal) models of human behavior [26, 21, 31], as well as on specific properties of robot behavior that aid collaboration [2, 14, 9]. However, to our knowledge, ours is the first work to analyze the optimal human assumption in the context of deep reinforcement learning, and to test potential solutions such as population-based training (PBT).

Choudhury et al. [7] is particularly related to our work. It evaluates whether it is useful to learn a human model using deep learning, compared to a more structured "theory of mind" human model. We are instead evaluating how useful it is to have a human model *at all*.

Multiagent reinforcement learning. Deep reinforcement learning has also been applied to multiagent settings, in which multiple agents take actions in a potentially non-competitive game [24, 15]. Some work tries to teach agents collaborative behaviors [22, 18] in environments where rewards are *not* shared across agents. The Bayesian Action Decoder [12] learns communicative policies that allow two agents to collaborate, and has been applied to the cooperative card game Hanabi. However, most multiagent RL research focuses on AI-AI interaction, rather than the human-AI setting.

Lerer and Peysakhovich [23] starts from the same observation that self-play will perform badly in general-sum games, and aims to do better given some data from agents that will be evaluated on at test time (analogous to our human data). However, they assume that the test time agents have settled into an equilibrium that their agent only needs to replicate, and so they train their agent with Observational Self-Play (OSP): a combination of imitation learning and MARL. In contrast, we allow for the case where humans do not play an equilibrium strategy (because they are suboptimal), and so we only use imitation learning to create human models, and train our agent using pure RL.

Imitation learning. Imitation learning [1, 16] aims to train agents that mimic the policies of a *demonstrator*. In our work, we use behavior cloning [30] to imitate demonstrations collected from humans, in order to learn human models to collaborate with. However, the main focus of our work is in the design of agents that can collaborate with these models, and not the models themselves.

3 Preliminaries

Multi-agent MDP. A multiagent Markov decision process [5] is defined by a tuple $\langle S, \alpha, \{A_{i\in\alpha}\}, \mathcal{T}, R\rangle$. S is a finite set of states, and $R : S \to \mathbb{R}$ is a real-valued reward function. α is a finite set of agents; A_i is the finite set of actions available to agent i. $\mathcal{T} : S\times A_1\times\cdots\times A_n\times S \to [0, 1]$ is a transition function that determines the next state given *all* of the agents' actions.

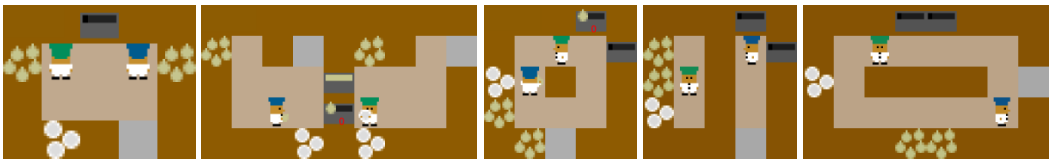


Figure 3: Experiment layouts. From left to right: *Cramped Room* presents low-level coordination challenges: in this shared, confined space it is very easy for the agents to collide. *Asymmetric Advantages* tests whether players can choose high-level strategies that play to their strengths, as illustrated in Figure 2. In *Coordination Ring*, players must coordinate to travel between the bottom left and top right corners of the layout. *Forced Coordination* instead removes collision coordination problems, and forces players to develop a high-level joint strategy, since neither player can serve a dish by themselves. *Counter Circuit* involves a non-obvious coordination strategy, where onions are passed over the counter to the pot, rather than being carried around.

Behavior cloning. One of the simplest approaches to imitation learning is given by behavior cloning, which learns a policy from expert demonstrations by directly learning a mapping from observations to actions with standard supervised learning methods [4]. Since we have a discrete action space, this is a traditional classification task. Our model takes an encoding of the state as input, and outputs a probability distribution over actions, and is trained using the standard cross-entropy loss function.

Population Based Training. Population Based Training (PBT) [19] is an online evolutionary algorithm which periodically adapts training hyperparameters and performs model selection. In multiagent RL, PBT maintains a population of agents, whose policies are parameterized by neural networks, and trained with a DRL algorithm. In our case, we use Proximal Policy Optimization (PPO) [32]. During each PBT iteration, pairs of agents are drawn from the population, trained for a number of timesteps, and have their performance recorded. At the end of each PBT iteration, the worst performing agents are replaced with copies of the best agents with mutated hyperparameters.

4 Environment and Agents

4.1 Overcooked

To test our hypotheses, we would like an environment in which coordination is challenging, and where deep RL algorithms work well. Existing environments have not been designed to be challenging for coordination, and so we build a new one based on the popular video game *Overcooked* [13], in which players control chefs in a kitchen to cook and serve dishes. Each dish takes several high-level actions to deliver, making strategy coordination difficult, in addition to the challenge of motion coordination.

We implement a simplified version of the environment, in which the only objects are onions, dishes, and soups (Figure 2). Players place 3 onions in a pot, leave them to cook for 20 timesteps, put the resulting soup in a dish, and serve it, giving all players a reward of 20. The six possible actions are: up, down, left, right, noop, and "interact", which does something based on the tile the player is facing, e.g. placing an onion on a counter. Each layout has one or more onion dispensers and dish dispensers, which provide an unlimited supply of onions and dishes respectively. Most of our layouts (Figure 3) were designed to lead to either low-level motion coordination challenges or high-level strategy challenges.

Agents should learn how to navigate the map, interact with objects, drop the objects off in the right locations, and finally serve completed dishes to the serving area. All the while, agents should be aware of what their partner is doing and coordinate with them effectively.

4.2 Human models

We created a web interface for the game, from which we were able to collect trajectories of humans playing with other humans for the layouts in Figure 3. In preliminary experiments we found that human models learned through behavior cloning performed better than ones learned with Generative Adversarial Imitation Learning (GAIL) [16], so decided to use the former throughout our experiments. To incentivize generalization in spite of the scarcity of human data, we perform behavior cloning over a manually designed featurization of the underlying game state.

For each layout we gathered ~16 human-human trajectories (for a total of 18k environment timesteps). We partition the joint trajectories into two subsets, and split each trajectory into two single-agent trajectories. For each layout and each subset, we learn a human model through behavior cloning. We treat one model, BC, as a human model we have access to, while the second model H_{Proxy} is treated as the ground truth human proxy to evaluate against at test time. On the first three layouts, when paired with themselves, most of these models perform similarly to an average human. Performance is significantly lower for the last two layouts (Forced Coordination and Counter Circuit).

The learned models sometimes got "stuck": they would perform the same action over and over again (such as walking into each other), without changing the state. We added a simple rule-based mechanism to get the agents unstuck by taking a random action. For more details see Appendix A.

4.3 Agents designed for self-play

We consider two DRL algorithms that train agents designed for self-play, and one planning algorithm.

DRL algorithms. For DRL we consider PPO trained in self-play (SP) and PBT. Since PBT involves training agents to perform well with a *population* of other agents, we might expect them to be more robust to potential partners compared to agents trained via self-play, and so PBT might coordinate better with humans. For PBT, all of the agents in the population used the same network architecture.

Planning algorithm. Working in the simplified Overcooked environment enables us to also compute near-optimal plans for the joint planning problem of delivering dishes. This establishes a baseline for performance and coordination, and is used to perform *coupled planning with re-planning*. With coupled planning, we compute the optimal joint plan for both agents. However, rather than executing the full plan, we only execute the first action of the plan, and then *replan* the entire optimal joint plan after we see our partner's action (since it may not be the same as the action we computed for them).

To achieve this, we pre-compute optimal joint motion plans for every possible starting and desired goal locations for each agent. We then create high-level actions such as "get an onion", "serve the dish", etc. and use the motion plans to compute the cost of each action. We use A^* search to find the optimal joint plan in this high-level action space. This planner does make some simplifying assumptions, detailed in Appendix E, making it near-optimal instead of optimal.

4.4 Agents designed for humans

We seek the simplest possible solution to having an agent that actually takes advantage of the human model. We embed our learned human model BC in the environment, treating it's choice of action as part of the dynamics. We then directly train a policy on this single-agent environment with PPO. We found empirically that the policies achieved best performance by initially training them in self-play, and then linearly annealing to training with the human model (see Appendix C).

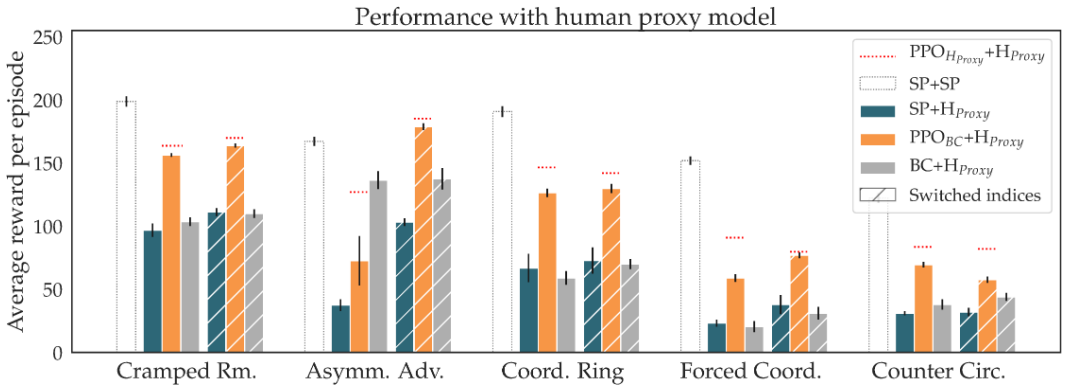
For planning, we implemented a model-based planner that uses hierarchical A^* search to act near-optimally, assuming access to the policy of the other player (see Appendix F). In order to preserve near-optimality we make our training and test human models deterministic, as dealing with stochasticity would be too computationally expensive.

5 Experiments in Simulation

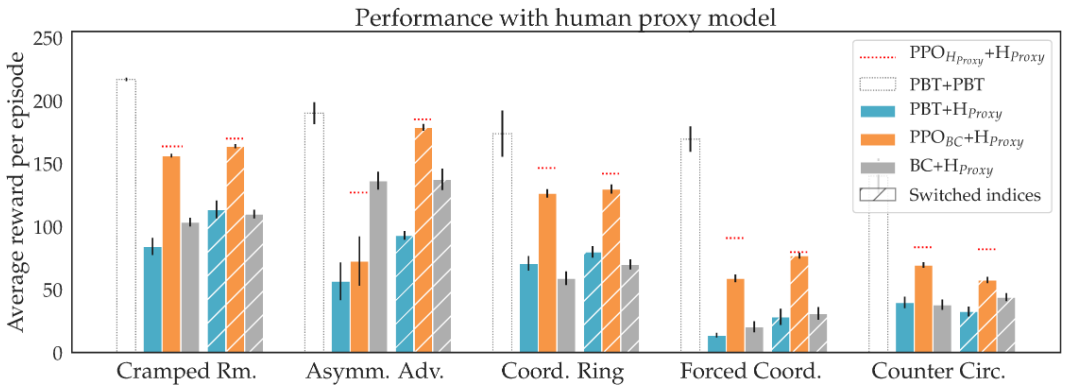
We pair each agent with the proxy human model and evaluate the team performance.

Independent variables. We vary the type of agent used. We have agents trained with themselves: self-play (SP), population-based training (PBT), coupled planning (CP); agents trained with the human model BC: reinforcement learning based (PPO$_{BC}$), and planning-based P$_{BC}$; and the imitation agent BC itself. Each agent is paired with H$_{Proxy}$ in each of the layouts in Figure 3.

Dependent measures. As good coordination between teammates is essential to achieve high returns in this environment, we use cumulative rewards over a horizon of 400 timesteps for our agents as a proxy for coordination ability. For all DRL experiments, we report average rewards across 100 rollouts and standard errors across 5 different seeds. To aid in interpreting these measures, we also compute a "gold standard" performance by training the PPO and planning agents not with BC, but with H$_{Proxy}$ itself, essentially giving them access to the ground truth human they will be paired with.



(a) Comparison with agents trained in self-play.



(b) Comparison with agents trained via PBT.

Figure 4: Rewards over trajectories of 400 timesteps for the different agents (agents trained with themselves – SP or PBT – in teal, agents trained with the human model – PPO$_{BC}$ – in orange, and imitation agents – BC – in gray), with standard error over 5 different seeds, paired with the proxy human H$_{Proxy}$. The white bars correspond to what the agents trained with themselves *expect* to achieve, i.e. their performance when paired with itself (SP+SP and PBT+PBT). First, these agents perform much worse with the proxy human than with themselves. Second, the PPO agent that trains with human data performs much better, as hypothesized. Third, imitation tends to perform somewhere in between the two other agents. The red dotted lines show the "gold standard" performance achieved by a PPO agent with direct access to the proxy model itself – the difference in performance between this agent and PPO$_{BC}$ stems from the innacuracy of the BC human model with respect to the actual H_{Proxy}. The hashed bars show results with the starting position of the agents switched. This most makes a difference for asymmetric layouts such as Asymmetric Advantages or Forced Coordination.

Analysis. We present quantitative results for DRL in Figure 4. Even though SP and PBT achieve excellent performance in self-play, when paired with a human model they struggle to even meet the performance of the imitation agent. There is a large gap between the imitation agent and gold standard performance. Note that the gold standard reward is lower than self-play methods paired with themselves, due to human suboptimality affecting the highest possible reward the agent can get. We then see that PPO$_{BC}$ outperforms the agents trained with themselves, getting closer to the gold standard. This supports our hypothesis that 1) self-play-like agents perform drastically worse when paired with humans, and 2) it is possible to improve performance significantly by taking the human into account. We show this holds (for these layouts) even when using an unsophisticated, behavior-cloning based model of the human.

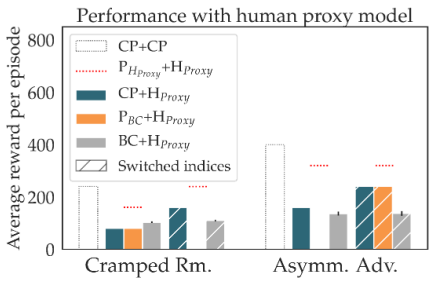


Figure 5: Comparison across planning methods. We see a similar trend: coupled planning (CP) performs well with itself (CP+CP) and worse with the proxy human (CP+H$_{Proxy}$). Having the correct model of the human (the dotted line) helps, but a bad model (P$_{BC}$+H$_{Proxy}$) can be much worse because agents get stuck (see Appendix G).

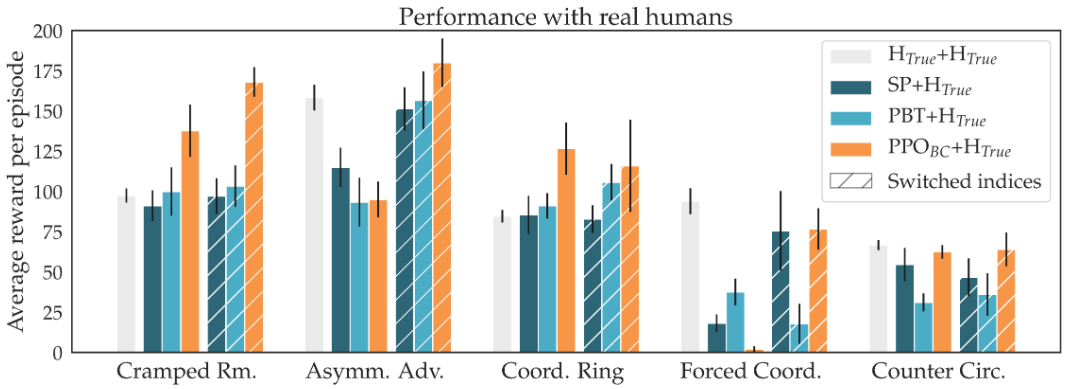


Figure 6: Average rewards over 400 timesteps of agents paired with real humans, with standard error across study participants. In most layouts, the PPO agent that trains with human data (PPO_{BC}, orange) performs better than agents that don't model the human (SP and PBT, teal), and in some layouts significantly so. We also include the performance of humans when playing with other humans (gray) for information. Note that for human-human performance, we took long trajectories and evaluated the reward obtained at 400 timesteps. In theory, humans could have performed better by optimizing for short-term reward near the end of the 400 timesteps, but we expect that this effect is small.

Due to computational complexity, model-based planning was only feasible on two layouts. Figure 5 shows the results on these layouts, demonstrating a similar trend. As expected, coupled planning achieves better self-play performance than reinforcement learning. But when pairing it with the human proxy, performance drastically drops, far below the gold standard. Qualitatively, we notice that a lot of this drop seems to happen because the agent expects optimal motion, whereas actual human play is much slower. Giving the planner access to the true human model and planning with respect to it is sufficient to improve performance (the dotted line above P_{BC}). However, when planning with BC but evaluating with H_{proxy}, the agent gets stuck in loops (see Appendix G).

Overall, these results showcase the benefit of getting access to a *good* human model – BC is in all likelihood closer to H_{Proxy} than to a real human. Next, we study to what extent the benefit is still there with a poorer model, i.e. when still using BC, but this time testing on real users.

6 User Study

Design. We varied the AI agent (SP vs. PBT vs. PPO_{BC}) and measured the average reward per episode when the agent was paired with a human user. We recruited 60 users (38 male, 19 female, 3 other, ages 20-59) on Amazon Mechanical Turk and used a between-subjects design, meaning each user was only paired with a single AI agent. Each user played all 5 task layouts, in the same order that was used when collecting human-human data for training. See Appendix H for more information.

Analysis. We present results in Figure 6. PPO_{BC} outperforms the self-play methods in three of the layouts, and is roughly on par with the best one of them in the other two. While the effect is not as strong as in simulation, it follows the same trend, where PPO_{BC} is overall preferable.

An ANOVA with agent type as a factor and layout and player index as covariates showed a significant main effect for agent type on reward ($F(2, 224) = 6.49$, $p < .01$), and the post-hoc analysis with Tukey HSD corrections confirmed that PPO_{BC} performed significantly better than SP ($p = .01$) and PBT ($p < .01$). This supports our hypothesis.

In some cases, PPO_{BC} also significantly outperforms human-human performance. Since imitation learning typically cannot exceed the performance of the demonstrator it is trained on, this suggests that in these cases PPO_{BC} would also outperform imitation learning.

We speculate that the differences across layouts are due to differences in the quality of BC and DRL algorithms across layouts. In Cramped Room, Coordination Ring, and the second setting of Asymmetric Advantages, we have both a good BC model as well as good DRL training, and so PPO_{BC} outperforms both self-play methods and human-human performance. In the first setting of Asymmetric Advantages, the DRL training does not work very well, and the resulting policy lets the human model do most of the hard work. (In fact, in the second setting of Asymmetric Advantages in

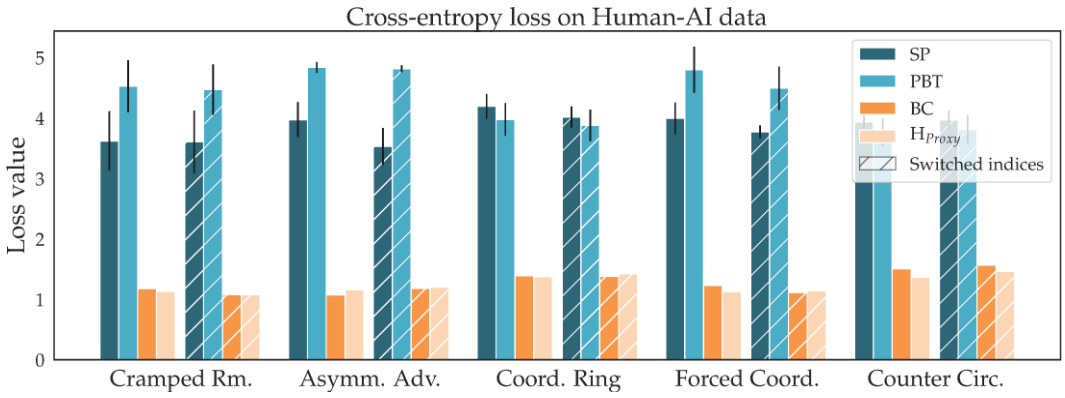


Figure 7: Cross-entropy loss incurred when using various models as a predictive model of the human in the human-AI data collected, with standard error over 5 different seeds. Unsurprisingly, SP and PBT are poor models of the human, while BC and H_{Proxy} are good models. See Appendix H for prediction accuracy.

Figure 4a, the human-AI team *beats* the AI-AI team, suggesting that the role played by the human is hard to learn using DRL.) In Forced Coordination and Counter Circuit, BC is a very poor human model, and so PPO$_{BC}$ still has an incorrect expectation, and doesn't perform as well.

We also guess that the effects are not as strong in simulation because humans are able to adapt to agent policies and figure out how to get the agent to perform well, a feat that our simple H_{Proxy} is unable to do. This primarily benefits self-play based methods, since they typically have opaque coordination policies, and doesn't help PPO$_{BC}$ as much, since there is less need to adapt to PPO$_{BC}$. We describe some particular scenarios in Section 7.

Figure 7 shows how well each model performs as a predictive model of the human, averaged across all human-AI data, and unsurprisingly finds that SP and PBT are poor models, while BC and H_{Proxy} are decent. Since SP and PBT expect the other player to be like themselves, they are effectively using a bad model of the human, explaining their poor performance with real humans. PPO$_{BC}$ instead expects the other player to be BC, a much better model, explaining its superior performance.

7 Qualitative Findings

Here, we *speculate* on some qualitative behaviors that we observed. We found similar behaviors between simulation and real users, and SP and PBT had similar types of failures, though the specific failures were different.

Adaptivity to the human. We observed that over the course of training the SP agents became very specialized, and so suffered greatly from distributional shift when paired with human models and real humans. For example, in Asymmetric Advantages, the SP agents only use the top pot, and ignore the bottom one. However, humans use both pots. The SP agent ends up waiting unproductively for the human to deliver a soup from the top pot, while the human has instead decided to fill up the bottom pot. In contrast, PPO$_{BC}$ learns to use both pots, depending on the context.

Leader/follower behavior. In Coordination Ring, SP and PBT agents tend to be very headstrong: for any specific portion of the task, they usually expect either clockwise or counterclockwise motion, but not both. Humans have no such preference, and so the SP and PBT agents often collide with them, and keep colliding until the human gives way. The PPO$_{BC}$ agent instead can take on both leader and follower roles. If it is carrying a plate to get a soup from the pot, it will insist on following the shorter path, even if a human is in the way. On the other hand, when picking which route to carry onions to the pots, it tends to adapt to the human's choice of route.

Adaptive humans. Real humans learn throughout the episode to anticipate and work with the agent's particular coordination protocols. For example, in Cramped Room, after picking up a soup, SP and PBT insist upon delivering the soup via right-down-interact instead of down-right-down-interact – even when a human is in the top right corner, blocking the way. Humans can figure this out and make sure that they are not in the way. Notably, PPO$_{BC}$ *cannot* learn and take advantage of human adaptivity, because the BC model is not adaptive.

8 Discussion

Summary. While agents trained via general DRL algorithms in collaborative environments are very good at coordinating with themselves, they are not able to handle human partners well, since they have never seen humans during training. We introduced a simple environment based on the game Overcooked that is particularly well-suited for studying coordination, and demonstrated quantitatively the poor performance of such agents when paired with a learned human model, and with actual humans. Agents that were explicitly designed to work well with a human model, even in a very naive way, achieved significantly better performance. Qualitatively, we observed that agents that learned about humans were significantly more *adaptive* and able to take on both *leader and follower roles* than agents that expected their partners to be optimal (or like them).

Limitations and future work. An alternative hypothesis for our results is that training against BC simply forces the trained agent to be robust to a wider variety of states, since BC is more stochastic than an agent trained via self-play, but it doesn't matter whether BC models real humans or not. We do not find this likely a priori, and we did try to check this: PBT is supposed to be more robust than self-play, but still has the same issue, and planning agents are automatically robust to states, but still showed the same broad trend. Nonetheless, it is possible that DRL applied to a sufficiently wide set of states could recoup most of the lost performance. One particular experiment we would like to run is to rain a single agent that works on arbitrary layouts. Since agents would be trained on a much wider variety of states, it could be that such agents require more general coordination protocols, and self-play-like methods will be more viable since they are forced to learn the same protocols that humans would use.

In contrast, in this work, we trained separate agents for each of the layouts in Figure 3. We limited the scope of each agent because of our choice to train the simplest human model, in order to showcase the importance of human data: if a naive model is already better, then more sophisticated ones will be too. Our findings open the door to exploring such models and algorithms:

Better human models: Using imitation learning for the human model is prone to distributional shift that reinforcement learning will exploit. One method to alleviate this would be to add inductive bias to the human model that makes it more likely to generalize out of distribution, for example by using theory of mind [7] or shared planning [17]. However, we could also use the standard data aggregation approach, where we periodically query humans for a new human-AI dataset with the current version of the agent, and retrain the human model to correct any errors caused by distribution shift.

Biasing population-based training towards humans: Agents trained via PBT should be able to coordinate well with any of the agents that were present in the population during training. So, we could train multiple human models using variants of imitation learning or theory of mind, and inject these human models as agents in the population. The human models need not even be accurate, as long as in aggregate they cover the range of possible human behavior. This becomes a variant of domain randomization [3] applied to interaction with humans.

Adapting to the human at test time: So far we have been assuming that we must deploy a static agent at test time, but we could have our agent adapt online. One approach would be to learn multiple human models (corresponding to different humans in the dataset). At test time, we can select the most likely human model [27], and choose actions using a model-based algorithm such as model predictive control [25]. We could also use a meta-learning algorithm such as MAML [11] to learn a policy that can quickly adapt to new humans at test time.

Humans who learn: We modeled the human policy as stationary, preserving the Markov assumption. However, in reality humans will be learning and adapting as they play the game, which we would ideally model. We could take this into account by using recurrent architectures, or by using a more explicit model of how humans learn.

Acknowledgments

We thank the researchers at the Center for Human Compatible AI and the Interact lab for valuable feedback. This work was supported by the Open Philanthropy Project, NSF CAREER, the NSF VeHICaL project (CNS-1545126), and National Science Foundation Graduate Research Fellowship Grant No. DGE 1752814.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[2] Rachid Alami, Aurélie Clodic, Vincent Montreuil, Emrah Akin Sisbot, and Raja Chatila. Toward human-aware robot task planning. In *AAAI spring symposium: to boldly go where no human-robot team has gone before*, pages 39–46, 2006.

[3] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[4] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]*, pages 103–129, Oxford, UK, UK, 1999. Oxford University. ISBN 0-19-853867-7. URL `http://dl.acm.org/citation.cfm?id=647636.733043`.

[5] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[6] Shan Carter and Michael Nielsen. Using artificial intelligence to augment human intelligence. *Distill*, 2017. doi: 10.23915/distill.00009. `https://distill.pub/2017/aia`.

[7] Rohan Choudhury, Gokul Swamy, Dylan Hadfield-Menell, and Anca Dragan. On the utility of model learning in HRI. 01 2019.

[8] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. OpenAI baselines. `https://github.com/openai/baselines`, 2017.

[9] Anca D Dragan, Kenton CT Lee, and Siddhartha S Srinivasa. Legibility and predictability of robot motion. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pages 301–308. IEEE Press, 2013.

[10] Douglas C Engelbart. Augmenting human intellect: A conceptual framework. *Menlo Park, CA*, 1962.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[12] Jakob N Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1811.01458*, 2018.

[13] Ghost Town Games. Overcooked, 2016. `https://store.steampowered.com/app/448510/Overcooked/`.

[14] Michael J Gielniak and Andrea L Thomaz. Generating anticipation in robot motion. In *2011 RO-MAN*, pages 449–454. IEEE, 2011.

[15] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

[17] M. K. Ho, J. MacGlashan, A. Greenwald, M. L. Littman, E. M. Hilliard, C. Trimbach, S. Brawner, J. B. Tenenbaum, M. Kleiman-Weiner, and J. L. Austerweil. Feature-based joint planning and norm learning in collaborative games. In A. Papafragou, D. Grodner, D. Mirman, and J.C. Trueswell, editors, *Proceedings of the 38th Annual Conference of the Cognitive Science Society*, pages 1158–1163, Austin, TX, 2016. Cognitive Science Society.

[18] Edward Hughes, Joel Z Leibo, Matthew Phillips, Karl Tuyls, Edgar Dueñez-Guzman, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin McKee, Raphael Koster, et al. Inequity aversion improves cooperation in intertemporal social dilemmas. In *Advances in Neural Information Processing Systems*, pages 3326–3336, 2018.

[19] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[20] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019. ISSN 0036-8075. doi: 10.1126/science.aau6249. URL `https://science.sciencemag.org/content/364/6443/859`.

[21] Shervin Javdani, Siddhartha S Srinivasa, and J Andrew Bagnell. Shared autonomy via hindsight optimization. *Robotics science and systems: online proceedings*, 2015.

[22] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[23] Adam Lerer and Alexander Peysakhovich. Learning existing social conventions via observationally augmented self-play. *AAAI / ACM conference on Artificial Intelligence, Ethics, and Society*, 2019.

[24] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[25] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[26] Stefanos Nikolaidis and Julie Shah. Human-robot cross-training: computational formulation, modeling and evaluation of a human team training strategy. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, pages 33–40. IEEE Press, 2013.

[27] Eshed Ohn-Bar, Kris M. Kitani, and Chieko Asakawa. Personalized dynamics models for adaptive assistive navigation interfaces. *CoRR*, abs/1804.04118, 2018.

[28] OpenAI. How to train your OpenAI Five. 2019. `https://openai.com/blog/how-to-train-your-openai-five/`.

[29] OpenAI. OpenAI Five finals. 2019. `https://openai.com/blog/openai-five-finals/`.

[30] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[31] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, volume 2. Ann Arbor, MI, USA, 2016.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[33] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[34] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

A Behavior cloning

We collected trajectories from Amazon Mechanical Turk. We removed trajectories that fell short of the intended human-human trajectory length of $T \approx 1200$ and very suboptimal ones (with reward roughly below to what could be achieved by one human on their own – i.e. less than 220 for Cramped Room, 280 for Asymmetric Advantages Ring, 150 for Coordination Ring, 160 for Forced Coordination, and 180 for Counter Circuit). After removal, we had 16 joint human-human trajectories for Cramped Room environment, 17 for Asymmetric Advantages, 16 for Coordination Ring, 12 for Forced Coordination, and 15 for Counter Circuit.

We divide the joint trajectories into two groups randomly, and split each joint trajectory $((s_0, (a_0^1, a_0^2), r_0), \ldots, (s_T, (a_T^1, a_T^2), r_T))$ into two single agent trajectories: $((s_0, a_0^1, r_0), \ldots, (s_T, a_T^1, r_T))$ and $((s_0, a_0^2, r_0), \ldots, (s_T, a_T^2, r_T))$. At the end of this process we have twice as many single-agent trajectories than the joint human-human trajectories we started with, for a total of approximately 36k environment timesteps for each layout.

We then use the two sets of single-agent human trajectories to train two human models, BC and H_{Proxy}, for each of the five layouts. We evaluate these trained behavior cloning models by pairing them with themselves averaging out reward across 100 rollouts with horizon $T = 400$.

In each subgroup, we used 85% of the data for training the behavior cloning model, and 15% for validation. To learn the policy, we used a feed-forward fully-connected neural network with 2 layers of hidden size 64. We report the hyperparameters used in Table 1. We run each experiment with 5 different seeds, leading to 5 BC and 5 H_{Proxy} models for each layout. We then manually choose one BC and one H_{Proxy} model based on the heuristic that the H_{Proxy} model should achieve slightly higher reward than the BC model – to make our usage of H_{Proxy} as a human proxy more realistic (as we would expect BC to underperform compared to the expert demonstrator).

Behavior cloning, unlike all other methods in this paper, is trained on a manually designed 64-dimensional featurization the state to incentivize learning policies that generalize well in spite of the limited amount of human-human data. Such featurization contains the relative positions to each player of: the other player, the closest onion, dish, soup, onion dispenser, dish dispenser, serving location, and pot (one for each pot state: empty, 1 onion, 2 onions, cooking, and ready). It also contains boolean values encoding the agent orientation and indicating whether the agent is adjacent to empty counters. We also include the agent's own absolute position in the layout.

To correct for a tendency of the learnt models to sometimes get stuck when performing low level action tasks, we added a hardcoded the behavior cloning model to take a random action if stuck in the same position for 3 or more consecutive timesteps. As far as we could tell, this does not significantly affect the behavior of the human models except in the intended way.

Behavior cloning hyperparameters					
Parameter	Cramped Rm.	Asym. Adv.	Coord. Ring	Forced Coord.	Counter Circ.
Learning Rate	1e-3	1e-3	1e-3	1e-3	1e-3
# Epochs	100	120	120	90	110
Adam epsilon	1e-8	1e-8	1e-8	1e-8	1e-8

Table 1: Hyperparameters for behavior cloning across the 5 layouts. *Adam epsilon* is the choice of the ϵ for the Adam optimizer used in these experiments.

B Self-play PPO

Unlike behavior cloning, PPO and other DRL methods were trained with a lossless state encoding consisting of 20 masks, each a matrix of size corresponding to the environment terrain grid size. Each mask contains information about a specific aspect of the state: the player's own position, the player's own orientation, location of dispensers of various types, location of objects on counters, etc.

In order to speed up training, we shaped the reward function to give agents some reward when placing an onion into the pot, when picking up a dish while a soup is cooking, and when picking up a soup

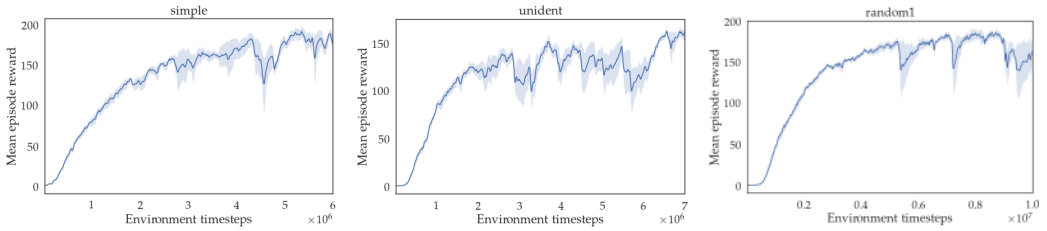
with a dish. The amount of reward shaping is reduced to 0 over the course of training with a linear schedule.

We parameterize the policy with a convolutional neural network with 3 convolutional layers (of sizes 5×5, 3×3, and 3×3 respectively), each of which has 25 filters, followed by 3 fully-connected layers with hidden size 32. Hyperparameters used and training curves are reported respectively in Table 2 and Figures 8.

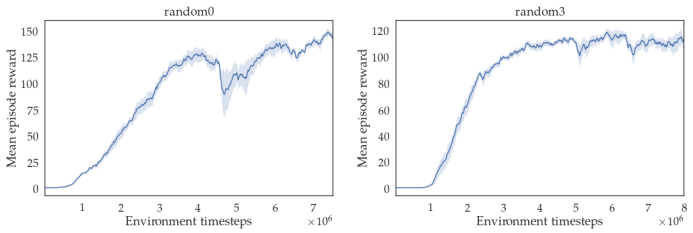
We use 5 seeds for our experiments, with respect to which we report all our standard errors.

PPO$_{SP}$ hyperparameters					
Parameter	Cramped Rm.	Asym. Adv.	Coord. Ring	Forced Coord.	Counter Circ.
Learning Rate	1e-3	1e-3	6e-4	8e-4	8e-4
VF coefficient	0.5	0.5	0.5	0.5	0.5
Rew. shaping horizon	2.5e6	2.5e6	3.5e6	2.5e6	2.5e6
# Minibatches	6	6	6	6	6
Minibatch size	2000	2000	2000	2000	2000

Table 2: Hyperparameters for PPO trained purely in self-play, across the 5 layouts. For simulation, we used 30 parallel environments. Similarly to the embedded human model case, parameters common to all layouts are: entropy coefficient $(= 0.1)$, gamma $(= 0.99)$, lambda $(= 0.98)$, clipping $(= 0.05)$, maximum gradient norm $(= 0.1)$, and gradient steps per minibatch per PPO step $(= 8)$. For a description of the parameters, see Table 3.



(a) Cramped Room **(b)** Asymmetric Advantages **(c)** Coordination Ring



(d) Forced Coordination **(e)** Counter Circuit

Figure 8: PPO$_{SP}$ self-play average episode rewards on each layout during training.

C PPO with embedded-agent environment

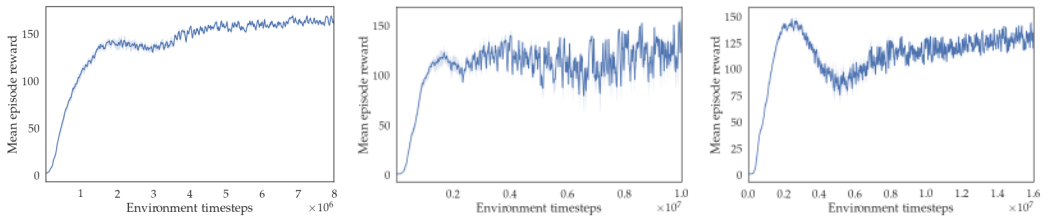
To train PPO with an embedded human model we use the same network structure as in the PPO$_{SP}$ case in Appendix B and similar hyperparameters, reported in Table 3. As in the PPO$_{SP}$ case, we use reward shaping and anneal it linearly throughout training.

Empirically, we found that – for most layouts – agents trained directly with competent human models to settle in local optima, never developing good game-play skills and letting the human models collect reward alone. Therefore, on all layouts except Forced Coordination, we initially train in pure self-play, and then anneal the amount of self-play linearly to zero, finally continuing training purely with the human model. We found this to improve the trained agents' performance. In Forced Coordination, both players need to learn game-playing skills in order to achieve any reward, so this problem doesn't occur.

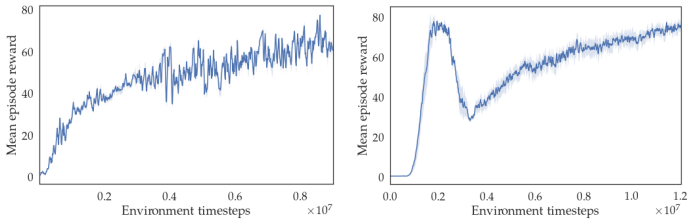
Training curves for the training (BC) and test (H_{Proxy}) models are reported respectively in Figures 9 and 10.

PPO_{BC} and $PPO_{H_{Proxy}}$ hyperparameters					
Parameter	Cramped Rm.	Asym. Adv.	Coord. Ring	Forced Coord.	Counter Circ.
Learning Rate	1e-3	1e-3	1e-3	1.5e-3	1.5e-3
LR annealing factor	3	3	1.5	2	3
VF coefficient	0.5	0.5	0.5	0.1	0.1
Rew. shaping horizon	1e6	6e6	5e6	4e6	4e6
Self-play annealing	[5e5, 3e6]	[1e6, 7e6]	[2e6, 6e6]	N/A	[1e6, 4e6]
# Minibatches	10	12	15	15	15
Minibatch size	1200	1000	800	800	800

Table 3: Hyperparameters for PPO trained on an embedded human model environment, across the 5 layouts. *LR annealing factor* corresponds to what factor the learning rate was annealed by linearly over the course of the training (i.e. ending at LR_0/LR_{factor}). *VF coefficient* is the weight to assign to the value function portion of the loss. *Reward shaping horizon* corresponds to the environment timestep in which reward shaping reaches zero, after being annealed lineraly. Of the two numbers reported for *self-play annealing*, the former refers to the environment timestep we begin to anneal from pure self-play to embedded human model training, and the latter to the timestep in which we reach pure human model embedding training. *N/A* indicates that no self-play was used during training. *# Minibatches* refers to the number of minibatches used at each PPO step, each of size *minibatch size*. For simulation, we used 30 parallel environments. Further parameters common for all layouts are: entropy coefficient ($= 0.1$), gamma ($= 0.99$), lambda ($= 0.98$), clipping ($= 0.05$), maximum gradient norm ($= 0.1$), and gradient steps per minibatch per PPO step ($= 8$). For further information, see the OpenAI baselines PPO documentation [8].



(a) Cramped Room **(b)** Asymmetric Advantages **(c)** Coordination Ring



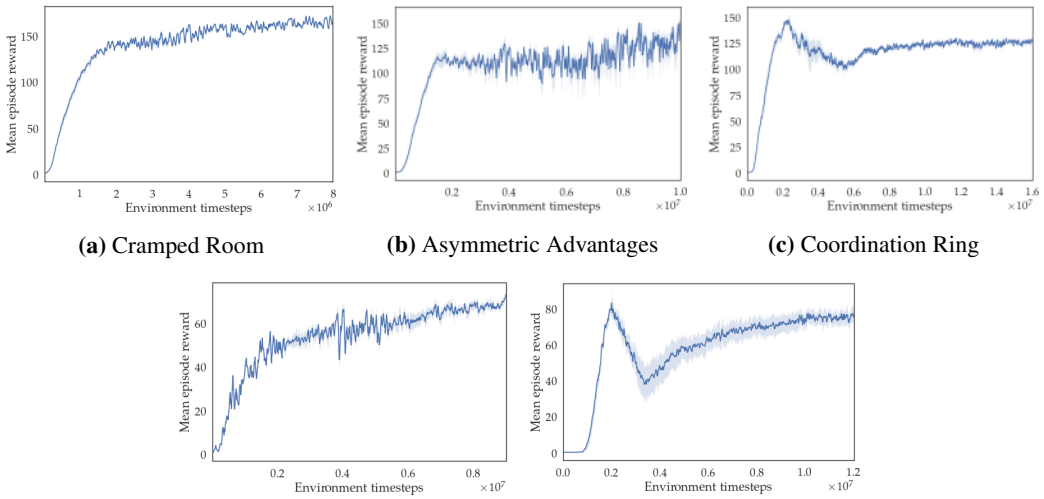
(d) Forced Coordination **(e)** Counter Circuit

Figure 9: PPO_{BC} average episode rewards on each layout during training over 400 horizon timesteps, when pairing the agent with itself or with BC in proportion to the current self-play annealing.

D Population Based Training

We trained population based training using a population of 3 agents, each of which is parameterized by a neural network trained with PPO, with the same structure as in C.

During each PBT iteration, all possible pairings of the 3 agents are trained using PPO, with each agent training on a embedded single-agent MDP with the other PPO agent fixed. PBT selection was conducted by replacing the worst performing agent with a mutated version of the hyperparameters. The parameters that could be mutated are lambda (initialized $= 0.98$), clipping (initialized $= 0.05$), learning rate (initialized $= 5e-3$), gradient steps per minibatch per PPO update (initialized $= 8$),



(a) Cramped Room **(b)** Asymmetric Advantages **(c)** Coordination Ring

(d) Forced Coordination **(e)** Counter Circuit

Figure 10: PPO$_{H_{Proxy}}$ average episode rewards on each layout during training over 400 horizon timesteps, when pairing the agent with itself or with H$_{Proxy}$ in proportion to the current self-play annealing.

entropy coefficient (initialized $= 0.5$), and value function coefficient (initialized $= 0.1$). At each PBT step, each parameter had a 33% chance of being mutated by either a factor of 0.75 or 1.25 (and clipped to the closest integer if necessary). For the lambda parameter, we mutate by $\pm\frac{\epsilon}{2}$ where ϵ is the distance to the closest of 0 or 1, to ensure that it will not go out of bounds.

As for the other DRL algorithms, we use reward shaping and anneal it linearly throughout training, and evaluate PBT reporting means and standard errors over 5 seeds. Hyperparameters and training reward curves are reported respectively in Table 4 and Figure 11.

PBT hyperparameters					
Parameter	Cramped Rm.	Asym. Adv.	Coord. Ring	Forced Coord.	Counter Circ.
Learning Rate	2e-3	8e-4	8e-4	3e-3	1e-3
Rew. shaping horizon	3e6	5e6	4e6	7e6	4e6
Env. steps per agent	8e6	1.1e7	5e6	8e6	6e6
# Minibatches	10	10	10	10	10
Minibatch size	2000	2000	2000	2000	2000
PPO iter. timesteps	40000	40000	40000	40000	40000

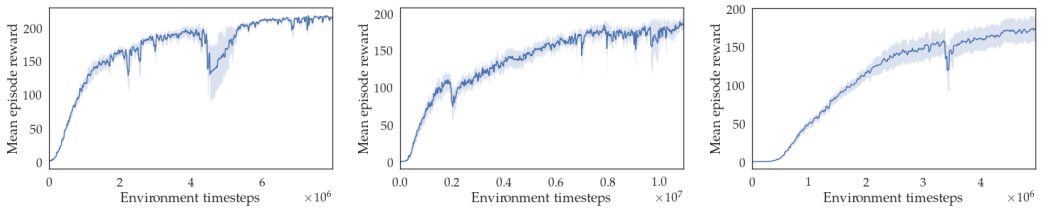
Table 4: Hyperparameters for PBT, across the 5 layouts. *PPO iteration timesteps* refers to the length in environment timesteps for each agent pairing training. For simulation, we used 50 parallel environments. The mutation parameters were equal across all layouts. For further description of the parameters, see Table 3.

E Near-optimal joint planner

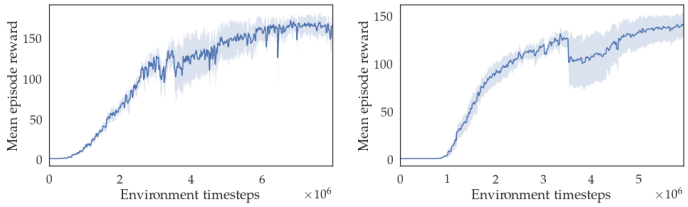
As mentioned in 4.3, to perform optimal planning we pre-compute optimal joint motion plans for every possible starting and desired goal location for each agent. This enables us to quickly query the cost of each motion plan when performing A^* search.

We then define the high-level actions: "get an onion", "serve the dish", etc, and map each joint high-level action onto specific joint motion plans. We then use A^* search to find the optimal joint plan in this high-level action space.

This planner does make some assumptions for computational reasons, making it near-optimal instead of optimal. In order to reduce the number of required joint motion plans by a factor of 16, we only consider position-states for the players, and not their orientations. This adds the possibility of wasting one timestep when executing certain motion plans, in order to get into the correct orientation. We



(a) Cramped Room **(b)** Asymmetric Advantages **(c)** Coordination Ring



(d) Forced Coordination **(e)** Counter Circuit

Figure 11: Average episode rewards on each layout during training for PBT agents when paired with each other, averaged across all agents in the population.

have added a set of additional conditions to check for such a case and reduce the impact of such an approximation, but they are not general.

Another limitation of the planning logic is that counters are not considered when selecting high level action, as it would increase the runtime of A^* by a large amount (since the number of actions available when holding an item would greatly increase). This is not of large importance in the two layouts we run planning experiments on. The use of counters in such scenarios is also minimal in human gameplay.

Another approximation made by the planner is only considering a 3 dish delivery look-ahead. Analyzing the rollouts, we would expect that increasing the horizon to 4 would not significantly impact the reward for the two layouts used in Section 5.

F Near-optimal model-based planner

Given a fixed partner model, we implement a near-optimal model-based planner that plays with the fixed partner. The planner uses a two-layered A^* search in which:

1) on a low level we use A^* in the game state space with edges being basic player joint actions (one of which is obtained by querying the partner model). To reduce the complexity of such search, we remove stochasticity from the partner model by taking the argmax probability action.

2) on a higher level we use A^* search in the state space in which edges are high level actions, similarly to those of the near-optimal joint planner described in Appendix E. Unlike in that case, it is unfeasible to pre-compute all low-level motion plans, as each motion plan does not only depend on the beginning positions and orientations of the agents, but also on other features of the state (since they could influence the actions returned by the partner model).

G Planning experiments

Due to computational constrains, when evaluating planning methods (such as in Figure 5), we evaluated on a horizon of 100 timesteps, and then multiplied by 4 to make the environment horizon comparable to all other experiments. This is another source of possible suboptimality in the planning experiments.

When observing Figure 5, we see that P_{BC}+H_{Proxy} performs much worse than the red dotted line (representing planning with respect to the actual test model H_{Proxy}). This is due to the fact that in the planning experiments all agents are set to choose actions deterministically (in order reduce the

planning complexity – as mentioned in appendix F), leading the agents to get often stuck in loops that last the whole remaining part of the trajectory, leading to little or no reward.

H Human-AI experiments

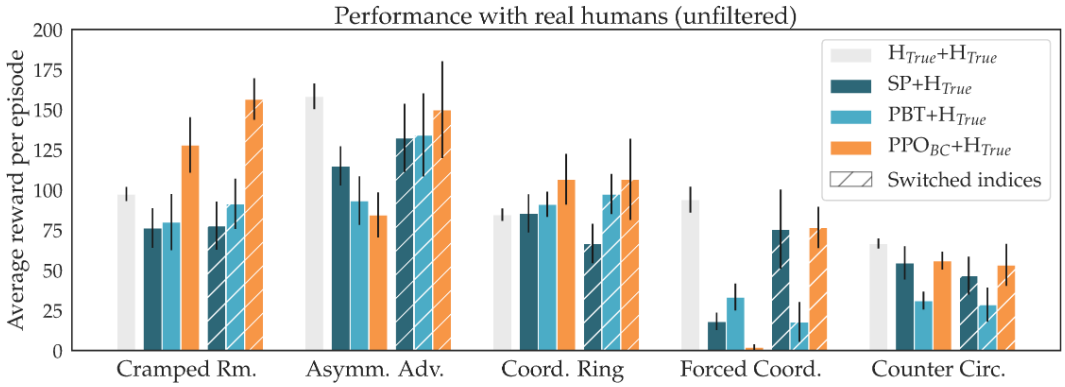


Figure 12: The results are mostly similar to those in Figure 6, with the exception of larger standard errors introduced by the non-cooperative trajectories. The reported standard errors are across the human participants for each agent type.

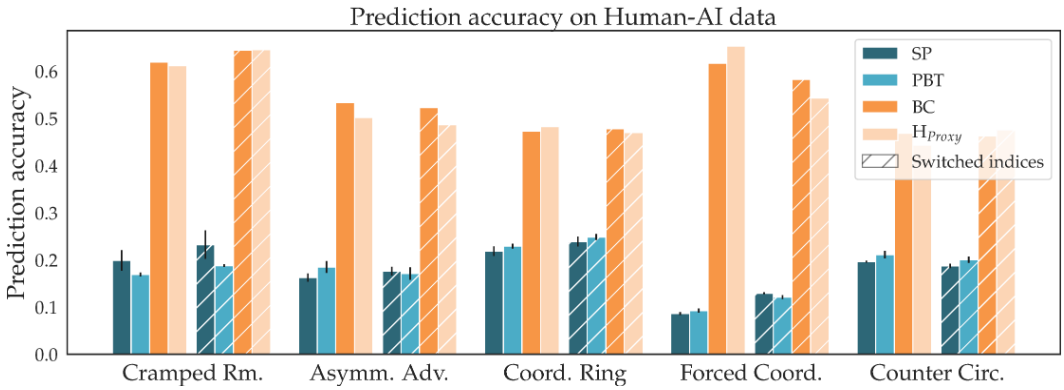


Figure 13: Accuracy of various models when used to predict human behavior in all of the human-AI trajectories. The standard errors for DRL are across the 5 training seeds, while for the human models we only use 1 seed. For each seed, we perform 100 evaluation runs.

The human-AI trajectories were collected with a horizon of 400 timesteps. The agents used in this experiment were trained with slightly different hyperparameters than those in the previous appendices: after we had results in simulation and from this user study, we improved the hyperparameters and updated the simulation results, but did not rerun the user study.

After manually inspecting the collected human-AI data, we removed all broken trajectories (human not performing any actions, and trajectories shorted than intended horizon). We also noticed that in a large amount of trajectories, humans were extremely non-cooperative, not trying to perform the task well, and mainly just observing the AI agent and interacting with it (e.g. getting in its way). Our hypothesis was that these participants were trying to "push the boundaries" of the AI agents and test where they would break – as they were told that they would be paired with AI agents. We also removed these non-cooperative trajectories before obtaining our results. Cumulatively across all layouts, we removed 15, 11, and 15 trajectories of humans paired with PBT, PPO$_{SP}$, and PPO$_{BC}$ agents respectively.

In Figure 12 we report human-AI performance without these trajectories removed. Performing an ANOVA on all the data with agent type as a factor and layout and player index as a covariates showed a significant main effect for agent type on reward ($F(2, 250) = 4.10$, $p < .01$), and the post-hoc analysis with Tukey HSD corrections confirmed that PPO$_{BC}$ performed significantly better than for PBT ($p < .01$), while fell just short of statistical significance for SP ($p = .06$).

We also report the results of using various agents as predictive models of human behavior in Figure 13. This is the same setup as in Figure 7, except we are reporting accuracies here instead of cross-entropy losses.

Another thing to note is that in order to obtain Figure 7, to prevent numerical overflows in the calculation of cross-entropy for the PBT and PPO$_{SP}$ models, we lower-bounded the probability outputted by all models for the correct action to $\epsilon = 1 \times 10^{-3}$. This reduces the loss of all models, but empirically affects PBT and PPO$_{SP}$ the most, as they are the models that are most commonly confidently-incorrect in predicting the human's action. We chose this ϵ as it is about 1 order of magnitude smaller than the smallest predictions assigned to the correct actions by the human models (i.e. the worst mistakes of the human models).