Team of Rivals: Hierarchical Deep Reinforcement Learning and Behavior Cloning for Multiplayer Poker

Avishag Shapira
Ben-Gurion University of the Negev
Be'er Sheva, Israel
shavish@post.bgu.ac.il

Ido Rom
Ben-Gurion University of the Negev
Be'er Sheva, Israel
romid@post.bgu.ac.il

Asaf Shabtai
Ben-Gurion University of the Negev
Be'er Sheva, Israel
shabtaia@bgu.ac.il

Gilad Katz
Ben-Gurion University of the Negev
Be'er Sheva, Israel
giladkz@bgu.ac.il

ABSTRACT

Multiplayer no-limit Texas Hold'em is considered a challenging benchmark for AI algorithms, due to the need for decision making under partial information, strategic deception, and non-stationary opponents. Classical equilibrium-based techniques do not extend cleanly to the multiplayer setting, and prevailing multiplayer solutions, such as LLMs, tend to be computationally intensive. This study introduces Havoc, a hierarchical deep RL approach that combines behavior cloning of individual human experts with a value-based master policy that selects, at each decision point, which specialist to deploy. By preserving distinct human play styles in the specialist policies and learning when to deploy them, Havoc adapts its strategy rapidly as table conditions shift. Despite limited training data, Havoc attains strong multiplayer performance, outperforming current state-of-the-art methods while also requiring substantially less computational resources.

KEYWORDS

deep reinforcement learning; behavioral cloning; poker

ACM Reference Format:
Avishag Shapira, Ido Rom, Asaf Shabtai, and Gilad Katz. 2026. Team of Rivals: Hierarchical Deep Reinforcement Learning and Behavior Cloning for Multiplayer Poker. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/LRNX6318

1 INTRODUCTION

Poker has long been considered a challenging problem in the field of AI. A learning model must operate under partial information, anticipate rival actions, and determine whether opponents are attempting deception (i.e., bluffs) [26]. Researchers in reinforcement learning have regarded poker as a central benchmark both due to its inherent difficulty and its relevance to other strategic domains such as bidding [31] and multi-agent collaboration [13].

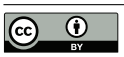
Over the past decade, AI systems for poker have made substantial progress [6, 8, 19], but most successes have been in heads-up (two-player) settings. Two-player formulations employ equilibrium-based strategies with performance guarantees [9]. In contrast, extending equilibrium methods to multiplayer games remains challenging in practice due to increased non-stationarity, strategic diversity, and the difficulty of scaling abstractions and search [9].

Applying deep reinforcement learning (DRL) to multiplayer poker is particularly difficult for two reasons. First, opponents can shift or alternate strategies within and across games, complicating convergence and inducing non-stationary training dynamics. Second, agents must both recognize and execute bluffing – apparently suboptimal actions taken to shape beliefs – which is intrinsically hard to learn from sparse, delayed rewards [26].

In this study, we present Hierarchical behAViOral Cloning (Havoc), a hierarchical DRL approach for multiplayer poker. The bottom tier comprises behavior-cloned (BC) specialists trained to imitate individual human experts, each preserving a distinct play style (e.g., aggressive, cautious, risk-seeking in specific contexts). The top tier is a value-based master agent that, at each decision point, selects which specialist acts next. By maintaining a portfolio of expert-aligned specialists and learning when to deploy them, the master policy can enact rapid, discrete strategy shifts as table conditions evolve. Our evaluation shows that Havoc outperforms state-of-the-art methods such as LLM-based approaches, despite being trained on a limited expert dataset and using substantially less computation overall. Our analysis further shows that the master policy frequently transitions between specialists during play, aligning with our core hypothesis that rapid, targeted switching among distinct styles is key to robust adaptation in multiplayer settings.

Our contributions are as follows:

- A hierarchical architecture for multiplayer poker that combines behavior cloning of individual human experts with a value-based master policy, enabling strategic use of diverse, style-preserving specialists.
- A state representation that integrates Markovian and non-Markovian information tailored to poker's imperfect-information dynamics, improving learning stability and convergence in practice.
- A modular, data-efficient framework that achieves strong multiplayer performance with limited expert data and substantially lower computation than recent large-scale systems,



Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/LRNX6318

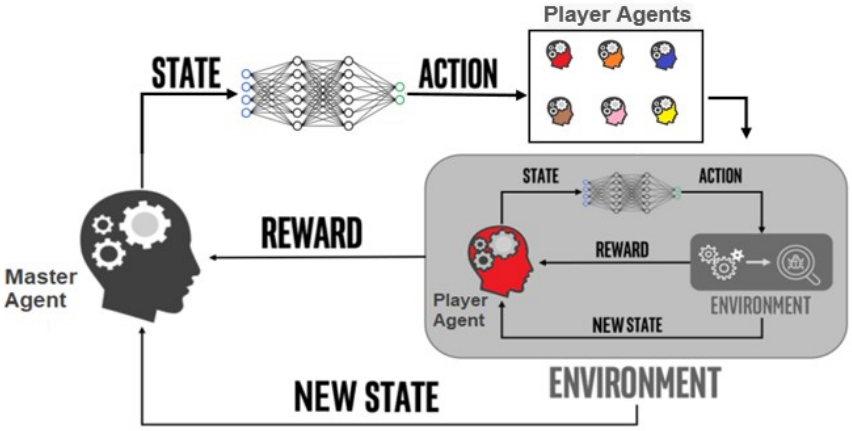


Figure 1: Havoc's two-tier architecture. The player Master agent evaluates the situation, and selects the Player agent that will play the current hand.

with potential applicability to other partial-information multi-agent settings (e.g., negotiation, bidding). The full implementation and supplementary material (including extended experiments and tables) are publicly available.[1]

2 RELATED WORK

2.1 Imitation Learning and Behavioral Cloning

Behavioral cloning (BC) is one of the main approaches of imitation learning [2]. Its goal is to mimic the actions of experts, thus removing the need to learn solely from interactions with the environment. In behavioral cloning (BC), the agent learns to directly imitate an expert policy from state–action demonstrations, typically by training a classifier or regressor to predict the expert's action given the observed state [2, 21]. Given high-quality demonstrations in sufficient quantity, BC models can obtain the optimal behavior even for complex continuous control tasks [22, 23].

Recent works in BC offer multiple advanced strategies. In [28], the authors propose BC from observation, which requires fewer post-demonstration environment interactions. The work by [22] demonstrates how to leverage noisy data (i.e., optimal and non-optimal examples from experts) for the training of an effective model. Adversarial BC, where samples are generated to reduce the amount of the required interaction with the environment, was proposed by [24].

Inverse reinforcement learning (IRL) is the problem of inferring the reward function that an expert agent is implicitly optimizing, given access to demonstrations or observed behavior [1]. In the domain of multi-agent systems, IRL has been proposed as an effective approach [20]. In [32], the authors showed that their algorithm is able to imitate expert behaviors in complex high-dimensional environments. In addition, their algorithm is able to learn reward functions that are highly correlated with the ground truth rewards.

2.2 Applying Machine Learning and DRL to Poker

Despite the great strides made in the field of artificial intelligence in recent years, poker remains a challenging problem [3]. This is the case because poker very elegantly captures the problems of missing information and uncertainty. A classic poker game has six players, with player aware of one's own cards and a variable number of public cards. A player has little to no information on the cards in the possession of the other five players and on the public cards that have not yet been revealed.

Due to the difficulty in applying AI to poker, most previous work in the field has reduced the problem to a two-player game [6, 8, 19]. All of these studies used varying types of decision tree algorithms. Additionally, all aforementioned algorithms utilized various forms of self-play, where the algorithm plays against older versions of itself in order to improve further. Another study on the heads-up poker problem (i.e., a two-player setup) is [27], where the authors present a new variant of Counterfactual Regret Minimization (CFR), that improves the performance of an inverse reinforcement learning (IRL) model. ReBel [7] combines Nash equilibrium with effective self-play and search for two-players setups. Another approach, proposed by [17] attempts to model opponents using LSTMs and Pattern Recognition Trees. AlphaHoldem [35] proposes the use of pseudo Siamese networks and self play to achieve SOTA results (and also perform well against human players).

We are aware of only two studies that apply AI algorithms to six-player poker game (while highly effective, we do not consider the use of LLM prompts [14, 16, 34] as an algorithm). In the first study [25], the authors used asynchronous self-play to train their model – an actor-critic architecture with an LSTM component to model the game history. The authors only report results against rule-based and Monte Carlo tree search-based algorithms, making the efficacy of the model difficult to assess. In [9], the authors propose an algorithm that is a type of CFR – an iterative self-learning algorithm that is initialized randomly, and slowly improves by playing against older versions of itself. To train the algorithm, the authors used two sets of abstractions, for the state and action spaces, respectively. The

[1] https://github.com/AvishagShapi/HAVOC

approach proved highly effective (it defeated expert human players), but computationally heavy: the algorithm was trained for 8 days on 64-core servers and used 12,400 CPU hours. Because Meta did not release the model, we cannot use it directly as a baseline in our study (but we do create a BC-based version for validation).

3 THE PROPOSED METHOD

3.1 Overview

Our proposed approach is presented in Figure 1. Havoc consists of a two-tier DRL architecture, with one *master agent* and multiple *player agents*. At every time step (i.e., when it's our agent's turn to play), the master agent analyzes the sequence of events that led to the current state, and then selects one of the available player agents. The chosen player agent also analyzes the sequence of events and performs a single action. Havoc then waits until it is again its turn.

We wish to point out two important aspects of our proposed approach. First, our master agent does not directly interact with the game. Its role is to analyze the state of the game and choose the "specialist" – the player the master deems most likely to lead to a win – that will make the next move. We argue that this approach is superior to a single model because it enables our model to more easily adapt its strategy to changing circumstances (see the analysis in Section 4.5).

The second aspect of Havoc we would like to point out is the fact that our approach could easily be adapted to changing environmental settings. The modular nature of our approach means that player agents can be updated without having to retrain the other agents, and that adding new player agents (i.e., changes to the master agent's action space) is likely to require only a limited fine-tuning of the master [4]. Havoc is, therefore, more dynamic and scalable compared to previously proposed approaches.

3.2 The State Space Representation

Our state representation consists of two sets of features: *Markovian* and *non-Markovian*. While the use of Markovian setups is more common in the literature, poker is inherently non-Markovian. This is the case because the way by which we reach the current state (i.e., the sequences of raises, calls, and folds) is as important as the state itself, if not more. Both of our agent types – master and player – utilize the same state representation. We now describe our feature subsets.

3.2.1 Markovian features subset. This set of features represents aspects of the game that are not dependent on previous turns. These features consist of the following:

(1) Number of active players.
(2) The current size of the pot (i.e., the amount of money "on the table").
(3) The cost to the agent to compare the current raise by playing a 'call'.
(4) The agent's position at the current round (a discrete variable with six possible values).
(5) The probability of winning *given the known information*.

The *probability of winning* feature mentioned above is another novelty proposed in this study. While previous studies [8, 9] used a large set of features to model the cards held by and visible to the player, we argue that this representation is not needed and is even counterproductive.

Our reasoning is based on two factors. First, modeling the various visible cards—as many as seven cards at the final stage of the game—requires a large feature space that makes training the neural net more challenging. Moreover, the neural net has to generalize various concepts regarding card combinations. For example, learning that two aces is good might generalize to a pair of kings, but not to a pair of queens. Secondly, we argue that *the cards themselves do not matter*. While human biases may place greater weight on some sets of cards, the only relevant consideration is the probability of winning with the current hand.

Based on the above reasoning, we substitute the representation of the cards visible to the player with a single scalar value denoting the probability of winning given the cards *currently visible to the player*. We obtain this value by running 1,000 simulations where we randomly assign values to all unknown players and common cards, and play each game to its conclusion. Our analysis shows that the probabilities of winning produced by these simulations are highly accurate. Running these simulations is computationally inexpensive (about one second per hand), and can be made more efficient through parallelization.

This representation enabled our model to converge faster and achieve a small improvement in performance. The use of this representation instead of the "standard" discrete representation of the cards reduces running time by about a third. The use of this feature also delivers a small improvement in performance (2% on average). Given that this average gain is too small to materially register in our reported metrics (e.g., with values on the order of 0.007, a 2% change falls below the resolution of the presented results), we report only the variant using the probability-of-winning feature to avoid redundancy. Importantly, in every comparison where our approach achieved a statistically significant improvement over the baselines, this held for both variants (with the feature and with the standard discrete representation).

3.2.2 Non-Markovian features subset. These features represent the actions taken by each player throughout the game. The feature set for any given time step in the game, therefore, is made up of discrete features – each representing a possible action – whose number is equal to the number of actions in the action space. In our experiments, the number of possible actions is five (see the next Section for details).

3.2.3 Using our feature sets to model the game. As explained above, the non-Markovian nature of poker requires that we model all the actions taken in the game, in the order in which they were played. Our state representation, therefore, must contain all this information rather than only the current state of the game. Moreover, we require a learning model that can accommodate inputs of various size, since the length of games may vary greatly.

We therefore employ a recurrent neural network (RNN)-based implementation of a DRL agent, for both types of agents: master and player. For the master, our chosen implementation is based on the DQRNN architecture [10, 15], which receives the two feature sets described above at each time step (see Figure 2).

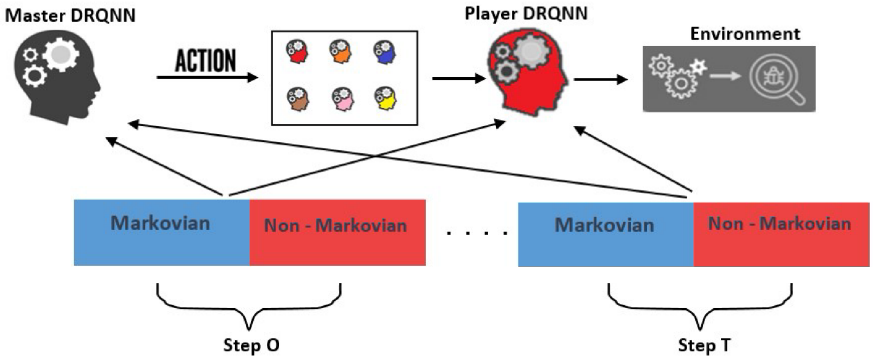


Figure 2: The state representation of our proposed approach. Both agents – Master and Player – receive the same set of features.

3.3 The Action Space

Given the small number of possible actions in poker, the action spaces of our two agents are fairly straightforward:

- **The player agent.** While poker consists of three actions – fold, call, and raise – the amount of money used when raising has significant effect on the outcome of the game. We follow the approach used in previous studies [9] and discretize the raise action into three different actions, determined by the amount of money being added. The ranges of values used in our actions were [1x-5x]/[6x-10x]/[≥ 10x][2] of the big blind or latest raise. Consequently, our player agent has five possible actions.
- **The master agent.** The master agent's only task is selecting the player agent that will play the next turn. Therefore, the number of actions available to the master is equal to the number of training set player agents.

3.4 Training the Player Agents

As explained in Section 1, we use behavioral cloning (BC) of individual human players to create player agents with different strategies and tactics. However, the main challenge in training these agents was overcoming the challenge of our limited training set (see Section 4.1 for details). Simply training a BC model on the limited dataset of each player's games produced poor results. Therefore, we devised a different training strategy.

Our approach was based on the understanding that, as in most games, experts tend to take similar actions in the majority of cases. What separates experts from non-experts (and experts from each other) is their actions in key stages of the game, where decisions are more complex and challenging [12]. Therefore, we apply a two-step training process:

- **General training** – first, we train one general behavior cloning model on our entire games dataset, regardless of player identities. Our aim is not to train a specialized agent, but rather to train a model that can learn the general play patterns of experts.
- **Fine-tuning** – in this step we create multiple copies of the BC model created in the previous step, and assign each copy to one of the human players whose behavior we wish to clone. We then further train each model on its assigned player's data, thus making the model's strategy more similar to that of its target. This additional training is generally known as fine-tuning, and it is common in NLP tasks [11] and transfer learning [30].

We use a standard LSTM architecture [33] to create our player agents, and use categorical cross-entropy as our loss function. The detailed architecture of the agents is described in Section 4.2.

3.5 Training the Master Agent

We train the master agent following the training of the individual player agents. This is necessary since the latter have to be stable for the former to learn to deploy them effectively. We use the DQRN architecture described in [10], with the specific architecture detailed in Section 4.2. For our reward function, we use the actual amount of money earned or lost at the end of each game. We chose this reward representation as it is the most accurate. Other, more regularized representations might have made our model too risk averse (i.e., the payoffs of large successful gambles would not have been fully delivered), or too reckless (when very large losses are not fully perceived).

4 EVALUATION

4.1 Training Datasets

Player agents training. We use the dataset released by [9], which consists of 10,000 games of six-player no-limit Texas hold'em poker, played by 13 human experts. After filtering players with an insufficient number of games, we had eight players who on average participated in 4,200 games each. Every move in the relevant games was converted into an individual state, leaving us with 67,034 states to train our BC models. Because the behavior cloning method used to train the player agents requires high quality games, the amount of data available to us is naturally limited. An important advantage of this dataset over others that are freely available is the fact that it contains information on all cards. Other datasets only share the cards that were made visible during the game—the cards played by the dealer, and those that the players expose when they call. Such limited information would have made behavioral cloning very difficult.

[2]Despite their prominence in popular culture, all-ins (one betting all of one's money) are very rare in professional poker. As such, they do not warrant an action of their own.

Table 1: The imitation accuracy of each player agent. This table shows the overall accuracy of each player agent in predicting the actions of its human player.

Player ID	player 1	player 2	player 3	player 4	player 5	player 6	player 7	player 8
Player Accuracy	81.68 %	83.8 %	82.88 %	84.4 %	80.27 %	86.77 %	83.5 %	85.84 %

Table 2: Imitation accuracy, broken down by action type. The present values were obtained by randomly sampling 1,066 states, played by a specific player

Action	Right Prediction	Sum of Actions	Percentage
Fold	560	571	98.1 %
Call	248	305	81.3 %
Raise	123	190	64.7 %
Raise-Call	163	190	85.85 %

Table 3: Hyperparameters for the DQRN master agent and the General Player BC model

DQRN (Master Agent)	
Architecture	LSTM (32 units) + Dense (Softmax)
Optimizer	Adam
Loss function	Mean Squared Error (MSE)
Discount Factor (γ)	0.99
Learning Rate	0.0001
Batch Size	256
Replay Memory Size	4000
Epsilon Decay	0.999946 (ϵ_{min} = 0.2)
General Player (BC Model)	
Architecture	LSTM (16 units) + Dense (Softmax)
Optimizer	Adam
Loss Function	Categorical Cross-Entropy
Learning Rate	0.0001
Batch Size	256

Master agent training. Unlike the player agents, which require expert data for training, the master agent can (and need to) be trained on a large set of diverse games. We randomly generate a large number of six-player poker games, randomly assign positions and cards to different players, and play each hand to its conclusion. In total, we generated 9,240 random hands to train our master agent.

4.2 Experimental Setup

The poker variant played is Texas Hold'em. Our DQRN model consists of an LSTM layer and a dense layer with softmax activation. We used the Adam optimizer, the Mean Squared Error (MSE) loss function, a learning rate of 0.0001, and a discount factor of 0.99. The rewards are given at the end of the game.

For the behavioral cloning models, we employ a two-stage training process: after training the general player model, the LSTM layer (first layer) is frozen, and only the dense layer is fine-tuned for each individual expert player using RMSprop optimizer. The detailed hyperparameters for both the DQRN master agent and the General Player BC model are presented in Table 3.

To ensure that our approach is evaluated on all possible game configurations, we randomize both Havoc's position at the table at each game, as well as the other five players that are chosen to participate. Training the master and player agents required approximately 10 hours on an NVIDIA Geforce GTX 1080 Ti with 11GB of memory.

Players are evaluated based on the amounts of money they won or lost throughout the tournament. Because each player plays a different number of games, we divide the wins by the number of played games, thus creating the *average win per-game* (AWG) metric, which we use in our evaluation.

4.3 Analyzing the Performance of the Player Agents

For our evaluation to be valid, it is essential that our player agents (which also act as our baselines) be able to clone the behavior of the human experts in a meaningful way, i.e., be good poker players. We begin by analyzing the overall accuracy of our player agents in imitating the human players. The results, presented in Table 1, show that our player agents' accuracy of predicting the right type of action—fold, call or raise—ranges between 80%-87%. It should be noted that the top-performing player agent in our experiments (Player 7) is not the one with the highest imitation accuracy.

Next we analyze the players' ability to predict specific action types. Because the most common action employed by expert players is fold, a player agent could potentially achieve a high prediction rate by always predicting the same action. We needed to ensure that our player agents that can accurately model all types of actions.

Our analysis is presented in Table 2. While our player agents are near perfect in predicting when to fold (often the easiest decision), they also achieve high accuracy in predicting when the human players chose to call (81.3%) and somewhat lower for raise (64.7%). While the percentages for the call and raise actions are lower than the fold action, it is important to note that a call action can often be considered as a raise (in both cases the player risks additional money), which is particularly true when previous players raised by large amounts, thus making a 'call' a significant risk. Moreover, the differences between a 'call' and small 'raise' might be interchangeable in human players' minds, which can contribute to the difficulty of accurately predicting their actions. For this reason, we added the line "raise-call" to Table 2, where we treat both actions as one. In this setup, Havoc's accuracy is significantly higher, reaching 85.85%.

4.3.1 Creating a Pluribus-based baseline. Finally, we used an unbiased estimator (of sorts) to evaluate the quality of our BC approach. While [9] did not release their model, Pluribus, *their released dataset contains its games*. Since Pluribus was reportedly able to reach superhuman performance, *we hypothesized that a behavioral cloning of this model would also yield superior results*.

We used the same process used to create our player agents, and fine-tune our model to create a Pluribus-based player: *BC-Pluribus*. We simulated 6,000 games in which BC-Pluribus was randomly matched each time with five other players. The results, presented in Table 4, show that BC-Pluribus outperforms all other BC models. Using the T statistical significance test, we determined that Pluribus' superior performance, compared to all the Player

Table 4: Average player earnings per game for all BC models, including BC-pluribus. These results do not include our proposed approach.

Player ID	player 1	player 2	player 3	player 4	player 5	player 6	player 7	player 8	**BC-Pluribus**
Earnings	-0.003	0.005	-0.001	-0.003	-0.017	0.002	0.003	-0.005	**0.007**
Std	0.0038	0.0045	0.0066	0.0094	0.0075	0.0034	0.0037	0.0075	0.0058

Table 5: Average player earnings per game (total earnings divided by the number of games played by each agent).

Player ID	**Havoc**	BC-Pluribus	player 1	player 2	player 3	player 4	player 5	player 6	player 7	player 8
Earnings	**0.013**	-0.007	-0.008	0.007	0.002	-0.007	-0.005	-0.01	0.012	-0.007
Std	0.0058	0.01	0.0038	0.0045	0.0066	0.001	0.0075	0.0034	0.0037	0.0075

agents, was significant with $p < 0.01$. These results clearly indicate that our two-phase approach – general training and fine-tuning – is indeed effective in creating proficient and specialized players. It should be noted that BC-Pluribus' imitation accuracy was 84.05%, in line with the other player agents.

While our Pluribus-based player achieved top results, we do not use it as one of the player agents the master can call upon. We do so for two reasons: first, Pluribus was trained using self-play and required a great deal of computation power. Our approach is more efficient – approximately 8 hours on a 48-core server compared to Pluribus' 12,400 hours – as it trains on comparatively little data and relies on leveraging human experts. As such, a comparison would not be fair. Secondly, because the Pluribus-based agent outperforms the human-based players, including it in our model would induce Havoc to only use this player and prevent it from combining the human players to create complex strategies. However, we do test Havoc's against BC-Pluribus, as it is a formidable baseline.

[Introduction] You are a Texas Holdem player, pre-flop. There are six players at the table. Each time I'll provide a state of a game and you should decide what the best action to do is.
[Action definition] The actions are: fold, call, raise1, raise2, raise3. We discretize the raise action into three different actions, determined by the amount of money being added. The ranges of values used in our actions were [1x-5x]/[6x-10x]/[10x<] of the big blind or latest raise. Consequently, our player agent has five possible actions.
[State definition] Every time it is your turn, I'll provide you the following information: 1. previous_actions: The last ten previous actions taken. **...** 2. who_plays: The players who are still in the game (based on their sitting location). **...** 3. money_on_table: The total amount bet in the hand so far. **...** 4. call_cost: The cost to call the current bet. **...** 5. place_in_table: The place of you (as a player) in the table. **...** 6. my_cards: Your cards. **...** 7. public_cards: The public cards. **...** 8. players_money: The amount of money each player has. **...**

Figure 3: Excerpts from the prompt used to train our ChatGPT-based poker agent. The annotations in blue were added for the reader's sake, and are not part of the prompt.

4.4 Evaluation Results

We designed a three-part evaluation process for our method. First, in Section 4.4.1 we compare our approach to BC-Pluribus, the agent we created by using behavioral cloning and fine-tuning on games played by the original Pluribus. Next, we conduct a leave-one-out (LOO) evaluation, where we use all player-agents but one to train Havoc, and compete against the remaining agent. The results of this evaluation are presented in Section 4.4.2. Finally, in Section 4.4.3 we use prompt engineering to compare Havoc to a ChatGPT-based poker player.

4.4.1 Comparison to Pluribus. As shown in Table 4, BC-Pluribus outperforms all the BC player agents that make up Havoc. While our BC-Pluribus is undoubtedly different from the original, these results confirm those reported in [9], and make Pluribus "the one to beat". We now compare Havoc's performance to that of Pluribus: we ran 4,000 games consisting of Havoc, Pluribus, and four other players randomly chosen from our pool of eight (positions at the table were assigned randomly). Note that while Havoc can utilize all eight player agents, it does not 'contain' Pluribus, nor did it play against it in its training phase.

The results of our evaluation are presented in Table 5. BC-Pluribus, which previously bested all agents by a wide margin, now loses to Havoc, with the latter achieving the top performance. Interestingly, some of our player agents (notably players 2 and 7) also improve their performance compared to the results reported in Table 4, despite the addition of another strong player (ours). We attribute this to the complex dynamics of the multi-agent environment, and plan to explore this further in future research. Using the T statistical test, we determined that Havoc performance is significantly better than that of all other agents (including BC-Pluribus, player 2, and player 7), with $p < 0.01$. In addition, even when BC-Pluribus is included in the specialist bank, Havoc does not over-specialize and still outperforms BC-Pluribus (see supplementary material for details).

4.4.2 Leave-one-out (LOO) evaluation. While BC-Pluribus is the best individual player, we also evaluate Havoc against each of the eight players that make it up. Because of the limited number of available players, we use a LOO evaluation: we train Havoc using seven player agents, run 4,000 games where Havoc and the eighth player are always at the table, together with four other random players. Again, positions at the table are assigned randomly. We repeat this process for all player agents.

The results of our evaluation are presented in Table 6. Due to space constraints, we only show the performance of Havoc and the eighth player (full results are presented in the supplementary material). Havoc significantly outperforms seven of the eight unfamiliar players, and does so by large margins. Our approach did not outperform player 2 in this evaluation, but it did obtain a positive score (i.e., earned money). This result show once again the complex nature and dynamics of multi-agent systems, especially since player 2 loses to BC-Pluribus, and Havoc beats the latter handily.

Table 6: Average player earnings per game for a LOO evaluation. Results are shown for Havoc and the player not included in the training set.

Player ID	player 1	player 2	player 3	player 4	player 5	player 6	player 7	player 8
Player Earnings	-0.002	**0.007**	-0.002	-0.009	-0.015	0.004	0.005	-0.005
Havoc Earning	**0.005**	0.005	**0.007**	**0.013**	**0.006**	**0.006**	**0.012**	**0.008**

Table 7: Average player earnings per game (total earnings divided by the number of games played by each agent) when evaluating Havoc against the ChatGPT-based player.

Player ID	**Havoc**	ChatGPT	player 1	player 2	player 3	player 4	player 5	player 6	player 7	player 8
Earnings	**0.012**	0.005	0.003	0.001	0	-0.009	-0.01	-0.009	0.002	-0.004
Std	0.0042	0.0045	0.0059	0.0062	0	0.0042	0.0047	0.0082	0.0017	0.0019

Table 8: Havoc's player agent selection strategy as a function of the number of active opponents remaining.

Player	1 opp.	2 opp.	3 opp.	4 opp.	5 opp.
player 1	0.9 %	1.87 %	0.33 %	0 %	0%
player 2	7.4 %	2.63 %	0.01 %	0 %	0%
player 3	0.01 %	45.73 %	53.83 %	0 %	0%
player 4	0 %	0 %	0 %	0 %	0%
player 5	0.44 %	7.84 %	0.47 %	0 %	0%
player 6	10.33 %	5.42 %	0.14 %	0 %	0%
player 7	76.98 %	24.56 %	6.34 %	91.2 %	99.05%
player 8	3.93 %	12.25 %	38.87 %	8.8 %	0.95%
Total	**100** %	**100** %	**100** %	**100** %	**100** %

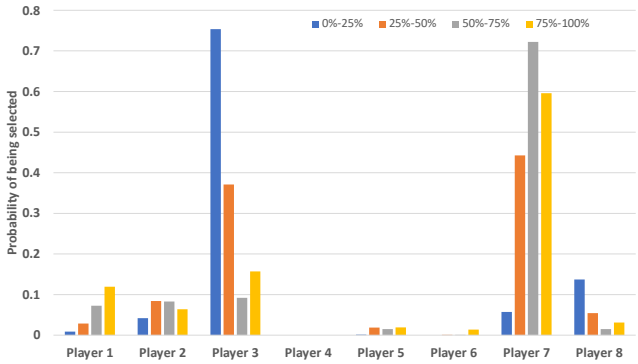


Figure 4: The probability of the master agent selecting a player agent, as a function of the former's assessment of the probability of winning the current game. The percentages of each color (i.e., probability range) sum up to 100%.

4.4.3 Comparison to a ChatGPT-based poker player. Large language models (LLMs) such as ChatGPT have proven highly effective in performing diverse and complex tasks, including passing the Bar exam [5], creating music [29], and rating Tai Chi routines [18]. Two recent studies demonstrate that ChatGPT is also a highly capable poker player. In [14], the author shows that the LLM-based model outperforms well-known baselines for two-person poker. The authors of [16] use prompt engineering to apply ChatGPT both to two-player and multi-player poker. In the two-player setup, their model outperformed two recent SOTA models, AlphaHoldem [35] and ReBel [7]. For the multi-player poker setup, the model performs well against a large number of bots. We now use the same approach for our evaluation. We build on the work of [16] and [14], and use ChatGPT to further evaluate our model in a multiplayer setting. We use ChatGPT 4.0, and provide it with the same information as our model (the information we provide is consistent with that of the works cited above). We also experimented with providing the specific cards instead of the probability of winning (as shown in the prompt excerpt in Figure 3), but this had no effect on the results. We ran 3,000 games consisting of Havoc, our ChatGPT player, and four other players randomly chosen from our pool of eight for each game (positions at the table were assigned randomly).

The results of our evaluation, presented in Table 7, show that while our ChatGPT player outperforms all individual player agents, Havoc is still the top performer by a large margin: 0.012 to 0.005 in favor of our approach. Using the T statistical significance test, we determined that the ChatGPT-based agent outperforms all agents except Havoc with $p < 0.01$. Havoc outperforms all other agents (including ChatGPT) with $p < 0.001$. Note that, as in our previous evaluations in Sections 4.4.1 and 4.4.2, we do not include the ChatGPT-based player in our training process and Havoc first encounters this player at the test time.

4.5 Analysis & Discussion

4.5.1 Player Agent Selection Distribution. Our goal is to determine whether our approach truly utilizes all the resources (i.e., players) at its disposal or focuses only on a subset. Table 8 presents the usage percentage of each player as a function of the number of active players in a game. The results show that while some players are more dominant than others, Havoc utilizes all players except for player 4. Interestingly, while player 7 is dominant in one opponent setups, it is far less used with two and three opponents. Player 7's dominance in the four- and five-opponent setup is largely irrelevant, because these situations are transitory and only happen at the beginning of games, before players start folding. The percentage of games consisting of more than three players after the first round (after all player had to call/raise or fold) is very small.

Next, we analyzed Havoc's selection strategy as a function of its assessment of *probability of winning* (see Section 3.2, which discusses the state space). The results of our analysis, presented in Figure 4, show that the probability of winning has a significant influence on Havoc's strategy. For example, Player 3 is very dominant when the chances of winning are low but is used sparingly when the probability increases. Moreover, while some players are more dominant than others, all players (with the exception of player 4) are all utilized by the master agent. This is a clear indication that Havoc creates a nuanced strategy that utilizes all available resources.

Game Stage	Winning Prob.	Pot Size	Call Cost	Num of Players	Fold	Call	Raise1	Raise2	Raise3
Pre_Flop	Medium	medium	low	4	0	0.0552	0.0541	0	0
River	high	low	low	2	0	0.0159	0.0577	0	0.0695
Flop	medium	medium	low	3	0	0.0392	0.0385	0	0
River	medium	medium	low	2	0	0.0321	0.0324	0	0
Turn	low	medium	low	2	0.0001	0.0245	0.023	0	0
River	low	medium	low	2	0	0.0204	0.0204	0	0
Pre_Flop	low	high	low	3	0.0198	0.0198	0	0	0
Flop	high	low	low	2	0	0.0163	0.0257	0	0.0178
Pre_Flop	medium	medium	low	3	0.0143	0.0326	0.008	0	0

Table 9: The normalized action variance of our eight player agents for various states. We present the top-10 states with the highest variance. The values of the state components are as follows: a) Game stage - pre-flop, flop, turn, river; b) Winning prob. - low, medium, high; c) Pot size - low, medium, high; d) Call cost - low, medium, high; e) Number of opponents - [2,5].

Table 10: The average number of times the master agent switched its chosen player agent, as a function of the number of its turn in the game.

# of Turns	2	3	4	5	6	7
# of Changes	0.765	1.121	2.236	2.642	3.75	4

4.5.2 Player Selection Duration. One of our main hypotheses in developing Havoc was that a hierarchical DRL model would be better able to quickly adapt its strategy to changing circumstances. Our reasoning was that a team of specialists will outperform a single generalist. To test the validity of this hypothesis, we analyzed the frequency by which Havoc alternates its chosen player agents throughout the game. A slow turnover, i.e., sticking with one "personality" for the majority of the game, will not support our hypothesis, while a quick turnover would.

Table 10 presents the average number of player agent switches as a function of the number of turns played in the game. Our approach changes its chosen player at a fast pace: for games in which Havoc took three actions or less, all chosen player agents were more likely than not to take a single action before being switched. For longer games (four to seven moves), the ratio of agent switches to game length decreases (usually because the probability of winning stops changing once all public cards are revealed), but the turnover remains high. These results clearly indicate that our DRL agent sees value in being to quickly alternate between our "specialists".

4.5.3 Differences in Player Agents' Styles. Our analysis makes it clear that the player agents employ different play styles. Nonetheless, it is interesting to identify the exact use-cases in which the agents differ. Our hypothesis was that expert players, like the ones we trained on in Section 3.4, mainly differ on the 'tough calls', while making similar decisions in easy-to-decide use cases.

To test our hypothesis, we analyzed the action distributions of our eight player agents across approximately 100k games. In each game, a random poker hand was generated, the game was paused at a random stage, and a random agent was selected. We recorded the state characteristics of this chosen agent, including stage, probability of winning, pot size, call cost, and number of players. We presented this state to all the player agents, and their chosen actions were recorded. After completing these games, we grouped states that share the same characteristics. For each group, we normalized the action distributions of each agent, ensuring that the sum of the distributions for all the agent's actions equaled one. Finally, we computed the standard deviation of these normalized distributions across the player agents for each action type separately.

The results, presented in Table 9, allow us to draw several conclusions. Firstly, the two actions with the most disagreements are 'Call' and 'Raise'. This is a common dilemma in poker games, and is clearly reflected in the data. Secondly, we see differences in the ways of pressing one's advantage: when the probability of winning is high, 'Call', 'Raise 1', and 'Raise 3' are all in use. This diversity reflects differences in style: some players only call or raise by small amounts to "lure the others in", while a very large raise is usually designed to push other players out and win the pot. Thirdly, even in cases when the probability of winning is low, some players will be aggressive and raise instead of calling or folding (usually when the number of active players is larger than two).

5 CONCLUSIONS

We present Havoc, a hierarchical DRL architecture that uses behavioral cloning to create a team of specialist player agents, with a value-based master that deploys them based on context for sharp, discrete strategy shifts and modular extensibility when adding or removing specialists. Beyond poker, the portfolio-of-specialists with a switching master has potential in diverse settings characterized by partial observability, strategic diversity, and non-stationarity, though this must still be validated. In electronic markets and strategic bidding, specialists can encode distinct risk regimes and microstructure responses while the master shifts stances across market regimes. In human–AI negotiation and multi-party dialogue, complementary negotiation styles can be selected as counterpart behavior evolves under incomplete information. In multi-robot or cyber-defense teams, style-specialized policies (e.g., patrol, intercept, deception) could be deployed adaptively as adversaries or conditions change, offering interpretable strategy changes with low recomputation cost, subject to safety and domain verification.

For future work, we plan to extend to additional domains (e.g., bidding, multi-party negotiations) and support dynamic incorporation of new specialists as novel opponents are encountered. Critically, we aim to learn temporally grounded opponent models—latent personality embeddings that update online—and condition both master selection and specialist behaviors on seated opponents, enabling deployment of the personalities that best exploit current table composition; our present model is identity-blind, so introducing robust, identity-aware opponent modeling under non-stationarity is a primary objective.

REFERENCES

[1] Stephen Adams, Tyler Cody, and Peter A Beling. 2022. A survey of inverse reinforcement learning. *Artificial Intelligence Review* 55, 6 (2022), 4307–4346.

[2] Michael Bain and Claude Sammut. 1995. A Framework for Behavioural Cloning.. In *Machine Intelligence 15*. 103–129.

[3] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. 2002. The challenge of poker. *Artificial Intelligence* 134, 1-2 (2002), 201–240.

[4] Yoni Birman, Shaked Hindi, Gilad Katz, and Asaf Shabtai. 2022. Cost-effective ensemble models selection using deep reinforcement learning. *Information Fusion* 77 (2022), 133–148.

[5] Michael Bommarito II and Daniel Martin Katz. 2022. GPT takes the bar exam. *arXiv preprint arXiv:2212.14402* (2022).

[6] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. 2015. Heads-up limit hold'em poker is solved. *Science* 347, 6218 (2015), 145–149.

[7] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. 2020. Combining deep reinforcement learning and search for imperfect-information games. *Advances in Neural Information Processing Systems* 33 (2020), 17057–17069.

[8] Noam Brown and Tuomas Sandholm. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359, 6374 (2018), 418–424.

[9] Noam Brown and Tuomas Sandholm. 2019. Superhuman AI for multiplayer poker. *Science* 365, 6456 (2019), 885–890.

[10] Clare Chen, Vincent Ying, and Dillon Laird. 2016. Deep q-learning with recurrent neural networks. *stanford cs229 course report* 4 (2016), 3.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] K Anders Ericsson, Michael J Prietula, and Edward T Cokely. 2007. The making of an expert. *Harvard business review* 85, 7/8 (2007), 114.

[13] Xiaohan Fang, Jinkuan Wang, Guanru Song, Yinghua Han, Qiang Zhao, and Zhiao Cao. 2019. Multi-agent reinforcement learning approach for residential microgrid energy scheduling. *Energies* 13, 1 (2019), 123.

[14] Akshat Gupta. 2023. Are ChatGPT and GPT-4 Good Poker Players?–A Pre-Flop Analysis. *arXiv preprint arXiv:2308.12466* (2023).

[15] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*.

[16] Chenghao Huang, Yanbo Cao, Yinlong Wen, Tao Zhou, and Yanru Zhang. 2024. PokerGPT: An End-to-End Lightweight Solver for Multi-Player Texas Hold'em via Large Language Model. *arXiv preprint arXiv:2401.06781* (2024).

[17] Xun Li and Risto Miikkulainen. 2018. Opponent modeling and exploitation in poker using evolved recurrent neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 189–196.

[18] Robert W McGee. 2023. Using Artificial Intelligence (AI) to Compose a Musical Score for a Tai Chi Tournament Routine: A ChatGPT Experiment. *Available at SSRN 4413424* (2023).

[19] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356, 6337 (2017), 508–513.

[20] Sriraam Natarajan, Gautam Kunapuli, Kshitij Judah, Prasad Tadepalli, Kristian Kersting, and Jude Shavlik. 2010. Multi-agent inverse reinforcement learning. In *2010 Ninth International Conference on Machine Learning and Applications*. IEEE, 395–400.

[21] Stéphane Ross and Drew Bagnell. 2010. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 661–668.

[22] Fumihiro Sasaki and Ryota Yamashina. 2020. Behavioral cloning from noisy demonstrations. In *International Conference on Learning Representations*.

[23] Fumihiro Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. 2018. Sample efficient imitation learning for continuous control. In *International conference on learning representations*.

[24] Fumihiro Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. 2020. Adversarial behavioral cloning. *Advanced Robotics* 34, 9 (2020), 592–598.

[25] Daming Shi, Xudong Guo, Yi Liu, and Wenhui Fan. 2022. Optimal policy of multiplayer poker via actor-critic reinforcement learning. *Entropy* 24, 6 (2022), 774.

[26] Finnegan Southey, Michael P Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. 2012. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411* (2012).

[27] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. 2015. Solving heads-up limit texas hold'em. In *Twenty-fourth international joint conference on artificial intelligence*.

[28] Faraz Torabi, Garrett Warnell, and Peter Stone. 2018. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954* (2018).

[29] Marcus Warnerfjord. 2023. Evaluating ChatGPT's Ability to Compose Music Using the MIDI File Format.

[30] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3, 1 (2016), 1–40.

[31] Yujian Ye, Dawei Qiu, Mingyang Sun, Dimitrios Papadaskalopoulos, and Goran Strbac. 2019. Deep reinforcement learning for strategic bidding in electricity markets. *IEEE Transactions on Smart Grid* 11, 2 (2019), 1343–1355.

[32] Lantao Yu, Jiaming Song, and Stefano Ermon. 2019. Multi-agent adversarial inverse reinforcement learning. In *International Conference on Machine Learning*. PMLR, 7194–7201.

[33] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation* 31, 7 (2019), 1235–1270.

[34] Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. 2024. Agent-pro: Learning to evolve via policy-level reflection and optimization. *arXiv preprint arXiv:2402.17574* (2024).

[35] Enmin Zhao, Renye Yan, Jinqiu Li, Kai Li, and Junliang Xing. 2022. AlphaHoldem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 4689–4697.