AlphaHoldem: High-Performance Artificial Intelligence for Heads-Up No-Limit Poker via End-to-End Reinforcement Learning

Enmin Zhao1,3*, Renye Yan3,1*, Jinqiu Li1,3, Kai Li1,3, Junliang Xing1,2,3†

1Institute of Automation, Chinese Academy of Sciences  2Tsinghua University  
3School of Artificial Intelligence, University of Chinese Academy of Sciences
zhaoenmin2018, yanrende2018, lijinqiu2021, kai.li@ia.ac.cn, jlxing@tsinghua.edu.cn

Abstract
Heads-up no-limit Texas hold’em (HUNL) is the quintessential game with imperfect information. Representative prior works like DeepStack and Libratus heavily rely on counterfactual regret minimization (CFR) and its variants to tackle HUNL. However, the prohibitive computation cost of CFR iteration makes it difficult for subsequent researchers to learn the CFR model in HUNL and apply it in other practical applications. In this work, we present AlphaHoldem, a high-performance and lightweight HUNL AI obtained with an end-to-end self-play reinforcement learning framework. The proposed framework adopts a pseudo-siamese architecture to directly learn from the input state information to the output actions by competing the learned model with its different historical versions. The main technical contributions include a novel state representation of card and betting information, a multi-task self-play training loss function, and a new model evaluation and selection metric to generate the final model. In a study involving 100,000 hands of poker, AlphaHoldem defeats Slumbot and DeepStack using only one PC with three days training. At the same time, AlphaHoldem only takes 2.9 milliseconds for each decision-making using only a single GPU, more than 1,000 times faster than DeepStack. We release the history data among among AlphaHoldem, Slumbot, and top human professionals in the author’s GitHub repository to facilitate further studies in this direction.

Introduction
Poker is a typical imperfect information game (IIG) that has a long history as a challenging problem for developing Artificial Intelligence (AI) that can address hidden information (Waterman 1970). Among different poker games, Heads-up no-limit Texas hold’em (HUNL) is a two-player poker game in which two cards are initially dealt face-down to each player, and additional cards are dealt with face-up in three subsequent rounds. No-limit means no restriction on the bet size, although it may be restricted by the total amount wagered in each game. Because of its explicit problem setting with large decision space (~10^{161} information sets) and strategic complexity, HUNL has been an excellent benchmark and challenging problem for developing AI algorithms for studying the two-player zero-sum games with imperfect information (Bard et al. 2013; Jackson 2013).

Recently, with the aid of increasing computing resources, computer programs have reached the performance that exceeds expert human players in many games, e.g., Go (Silver et al. 2016), MahJong (Li et al. 2020), DOTA (Berner et al. 2019), and StarCraft (Vinyals et al. 2019). These AI systems collect a tremendous amount of replay samples either from human experts or self-play of the system to train some complex learning models. The adoption of deep neural networks significantly improves the learning ability and performance of these systems. However, the model training process often lasts for dozens of days using thousands of CPU/GPUs, making these models extremely expensive to obtain. According to the reported computing resources used in AlphaGo (Silver et al. 2016), training an AlphaGo model costs about 35 million dollars.

As for the HUNL AI, new algorithms are also progressing very fast under the counterfactual regret minimization (CFR) framework (Zinkevich et al. 2007). DeepStack (Moravcik et al. 2017) and Libratus (Brown and Sandholm 2018) are independently developed and demonstrate expert-level performance. Both DeepStack and Libratus compute an abstraction of the game and introduce sub-game solving with the CFR+ (Tammelin et al. 2015) algorithm to learn HUNL AIs. Under the CFR framework, the primary computation cost comes from the CFR iteration process performed in both the model training and testing stages. To ensure high-quality prediction, this iteration process often needs to be carried out for more than 1,000 times in practice (Moravcik et al. 2017). This restriction makes the training of a high-performance CFR-based HUNL AI computationally infeasible for most research institutions and prevents the application of the CFR model into larger IIGs.

This work aims to develop a high-performance HUNL AI with affordable computation and storage costs for small research institutions and inspire further studies to develop more universal solutions for AI of Texas hold’em and other IIGs. To this end, we propose AlphaHoldem, a HUNL AI trained from an end-to-end reinforcement learning framework rather than the CFR framework that gives birth to most of the current HUNL AIs. We design a pseudo-Siamese architecture in this framework that directly learns from the input game information to produce the action through a single
feedforward pass of a neural network. This new architecture eliminates the need for highly computationally intensive CFR iterative inference during training and testing stages. To accelerate the training process of AlphaHoldem, we develop a set of new techniques for efficient learning the AlphaHoldem framework, including game state representations, training loss functions, and model generation strategies.

In contrast to previous abstraction-based methods in HUNL AI design, AlphaHoldem does not perform any card information abstractions using human domain knowledge. Instead, it encodes the game information into tensors containing the current and historical poker information. This new multidimensional tensor representation permits efficient learning of the decision model using convolutional networks. As for the learning algorithms, we propose a new loss function in the actor-critic paradigm which significantly improves the model learning speed and stability. To perform better early-stopping and generate a strong HUNL model, we propose a new self-play procedure to simultaneously reduce the training cost and guarantee the model performance. This is achieved by keeping only one agent as the main training objective but maintains a pool of competing agents to play with the main agent to ensure the replay sampling diversity. The proposed new loss function also helps in selecting the competing agents in the pool.

The size of the whole AlphaHoldem model is less than 100MB. We finish the training of the AlphaHoldem AI in three days using only one single computing server of 8 GPUs and 64 CPU cores. During inference, AlphaHoldem takes only $2.9 \times 10^{-3}$ seconds for each decision in a NVIDIA TITAN V GPU. We evaluate the effectiveness of AlphaHoldem through extensive experimental analyses and comparisons. In a study involving 200,000 hands of poker, AlphaHoldem beats DeepStack and Slumbot with statistical significance by a margin of 16.91 mbb/h and 111.56 mbb/h, respectively. This work makes the following three main contributions:

- We present a general and end-to-end self-play reinforcement learning framework to tackle the challenging HUNL problem: inference from state information directly to the final action using only a forward pass of the neural network in each decision point.
- We develop a set of new techniques to speed up the learning process of the AlphaHoldem: a new game state representation without the abstraction of the card information or any human knowledge, a new policy loss function that limits the distribution of policies, and a new self-play procedure that quickly generates the best model.
- We obtain a high-performance HUNL AI AlphaHoldem: it is trained in three days using a single machine and beats the current two best HUNL AI, Slumbot and DeepStack, with only 3ms decision time, more than 1,000 times faster than DeepStack.

We have released the history data among AlphaHoldem, Slumbot, and top human professionals for research purposes in the author’s GitHub repository\textsuperscript{1} to facilitate further studies in large-scale IIGs.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
& Train Resources & Test Resources & \\
& CPU hours & GPU hours & Storage Resources & \\
\hline
DeepStack & $1.53 \times 10^{6}$ & $1.31 \times 10^{4}$ & 16MB & a GPU \\
Libratus & $> 3 \times 10^{6}$ & 0 & > 100GB & a GPU \\
\hline
AlphaHoldem (CPU Version) & $5 \times 10^{4}$ & 210 & 98MB & a CPU \\
AlphaHoldem (GPU Version) & $4 \times 10^{3}$ & 580 & 98MB & a GPU \\
\hline
\end{tabular}
\caption{Cost comparisons of HUNL AIs. AlphaHoldem achieves good results with less computational resources.}
\end{table}

\section*{Related Work}

Texas hold' em has long served as the benchmarks for developing IIG algorithms (Rubin and Watson 2011; Bard et al. 2013). Most early studies are heuristic-based methods (Beattie et al. 2007), and the AI performance based on them are relatively weak. In 2007, the seminal counterfactual regret minimization (CFR) (Zinkevich et al. 2007) algorithm is introduced to efficiently solving two-player zero-sum IIGs. CFR is a conceptually simple iterative algorithm that tries to minimize the regrets of both players so that the time-averaged strategy approach to the Nash equilibrium. Thereafter, CFR-based methods dominate the design of Texas hold'em AI (Lanctot et al. 2009; Jackson 2013; Burch, Johanson, and Bowling 2014). After the Head-up Limit Texas hold'em is solved in 2015 (Bowling et al. 2015), much research effort has focused on No-limit Texas hold'em and recently made milestone progress. DeepStack (Moravcik et al. 2017) adopts a neural network to approximate the tabular CFR and performs recursive reasoning. Libratus (Brown and Sandholm 2018) computes a blueprint for the overall strategy and fixes potential weaknesses identified by the opponents in the blueprint strategy. They are independently developed and both have defeated professional human players in HUNL. Pluribus (Brown and Sandholm 2019b) further applies similar procedure into multiplayer no-limit Texas hold'em and report super-human performance.

Despite significant progress, all the milestone Texas hold'em AIs are built upon CFR, which requires costly computation to obtain the counterfactual values and large storage to store the model. In the inference stage, the CFR iteration process also consumes much computation. Besides, these methods only solve an abstracted game employ different kinds of Texas hold'em domain knowledge. This work aims to overcome these limitations of current HUNL AIs and produce a more general solution. Some recent works also make efforts towards this direction. NFSP (Heinrich and Silver 2016) and Poker-CNN (Yakovenko et al. 2016) have approached state-of-the-art performance in limit Texas hold’em. DeepCFR (Brown et al. 2019) further improves the performance by approximates CFR’s behavior in the game using deep neural networks and Discounted CFR (Brown and Sandholm 2019a). Inspired by AlphaGo (Silver et al. 2016), ReBel (Brown et al. 2020) combines search and reinforcement learning in HUNL AI. Despite superhuman performance reported, it still needs iterative learning in both the

\textsuperscript{1}https://github.com/ZhaoEnMin/AlphaHoldem_Data/
**Texas Hold’em Rules.** Texas hold’em is a repeated game, each of which begins with two cards (hole cards) dealt face down to each player, and then five cards (community cards) dealt face up in three stages. The stages consist of a series of three cards (the flop), later an additional single card (the turn), and a final card (the river). Each player seeks the best five cards from any combination of the five community cards and two hole cards. Players have betting options to check, call, raise, or fold. Rounds of betting take place before the flop is dealt with and after each subsequent deal. The player who has the best hand and has not folded by the end of all betting rounds wins all the money bet for the hand, known as the pot. In HUNL, two players play the game with the bet size restricted only by the total amount wagered in each game. Figure 1(a) illustrates one HUNL game, and Figure 1(b) shows the cards strength.

**Reinforcement Learning (RL).** In self-play, given a fixed opponent, the original two-player HUNL game reduces to a single-player RL problem since the opponent can be regarded as part of the environment. We consider the standard RL formalism, i.e., Markov Decision Process (MDP). An MDP consists of a set of states \( S = \{s_0, s_1, s_2, \ldots, s_t, \ldots\} \), a set of actions \( \mathcal{A} = \{a_k\}_{k=1}^{K} \), and a reward function \( r: S \times \mathcal{A} \to \mathbb{R} \). After executing an action \( a_t \in \mathcal{A} \) at each state \( s_t \in S \), the agent will enter a new state \( s_{t+1} \) according to the transition probability model and get a reward \( r(s_{t+1}|s_{t}, a_t) \). The objective of the agent is to maximize the cumulative rewards \( R = \sum_{t=0}^{\infty} \gamma^t r(s_{t+1}|s_{t}, a_t) \), where \( \gamma \) is the discount factor to favor more recent rewards.

**AlphaHoldem Architecture.** AlphaHoldem aims to remove the expensive computation of CFR iteration in both the training and testing stages of a HUNL AI. It thus pursues an end-to-end learning framework to perform efficient and effective decision-making in IIGs. Here end-to-end means that the framework directly accepts the game board information and outputs the actions without encoding handcrafted features as inputs or performing iterative reasoning in the decision process. AlphaHoldem adopts the RL framework to achieve this goal, and the only force to drive the model to learn is the game reward.

In HUNL, the game board information includes the current and historical card information and the player action information. The agent chooses from a set of betting actions to play the game and try to win more rewards. To capture the complex relationship among the game board information, the desired betting actions, and the game rewards, AlphaHoldem designs a pseudo-Siamese architecture equipped with the RL schema to learn the underlying relationships from end to end. We illustrate the end-to-end learning architecture of AlphaHoldem in Figure 2.

As shown in Figure 2, the input of the architecture is the game state representations of action and card information, which are respectively sent to the top and bottom streams of the Siamese architecture. Since the action and card representations provide different kinds of information to the learning architecture, we first isolate the parameter-sharing of the Siamese architecture to enable the two ConvNets to learn adaptive feature representations, which are then fused through fully connected layers to produce the desired actions. This design is the reason why we call it pseudo-Siamese architecture. To train this deep architecture, we present a novel Trinal-Clip loss function to update the model parameters using off-policy RL algorithms. We obtain the final model through a new self-play procedure that plays the current model with a pool of its \( K \) best historical versions to sample diverse training data from the huge game state space.
Learning Speedup Techniques

The core to the success of AlphaHoldem depends on a set of training speedup techniques that makes the learning of a superhuman HUNL AI with the current lowest computation and storage costs possible. In the following, we highlight and expatiate three new crucial techniques in speedup training the AlphaHoldem model. We believe these new techniques and underlying principles are helpful to develop general learning algorithms for more IIG AIs.

Effective Game State Representation

The existence of private information and flexibility of bet size cause the HUNL AI learning extremely challenging. Previous CFR-based methods often abstract the cards and bet information into different groups and use their concatenated coding vectors as game state representation to make the iterative reasoning process feasible. The abstracted code vectors lose many important game information and may not capture the complex relationship between the game information and optimal decisions. To obtain an effective and suitable feature representation for end-to-end learning from the game state directly to the desired decision, we design a new multidimensional feature representation to encode both the current and historical card and bet information.

In HUNL, the card information and action information exhibit different characteristics. We thus represent them as two separated three-dimension tensors and let the following network learn to fuse them (Figure 2). We design the card tensor in six channels to represent the agent’s two hole cards, three flop cards, one turn card, one river card, all public cards, and all hole and public cards. Each channel is a $4 \times 13$ sparse binary matrix, with 1 in each position denoting the corresponding card. For the actor tensor, since there are usually at most six sequential actions in each of the four rounds, we design it in 24 channels. Each channel is a $4 \times n_h$ sparse binary matrix, where $n_h$ is the number of betting options, and the four dimensions correspond to the first player’s action, the second player’s action, the sum of two players’ action, and the legal actions. To understand this representation, Figure 3 illustrates one example that a player in the small blind plays an action ‘bet pot’ after getting a hand ‘AsAc’.

This representation has several advantages: 1) there is no abstraction of the card information thus reserves all the game information; 2) the action representation is general and can denote different number of betting options (though $n_h = 9$ produce satisfactory results in the experiment); 3) all the historical information is encoded to aid reasoning with hidden information; and 4) the multidimensional tensor representation is very suitable for modern deep neural architectures like ResNet (He et al. 2016) to learn effective feature hierarchies, as verified in the AlphaGo AI training.

Effective Learning with Trinal-Clip PPO

With the multidimensional feature representation, one key factor to train the deep architecture is the learning paradigm with suitable loss functions. We adopt the actor-critic paradigm with off-policy training (Konda and Tsitsiklis 2000), which performs updating asynchronously on replayed experiences. The actor-critic paradigm trains a value function $V_\phi(s_t)$ and a policy $\pi_\theta(a_t|s_t)$, and updates them iteratively by sampling from the replay buffer.

We employ the popular Proximal Policy Optimization (PPO) (Schulman et al. 2017) learning algorithm to update the policies $\pi_\theta$ in the actor-critic framework. PPO defines the ratio function $r_t(\theta) = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_{\theta}(a_t|s_t)}$ as the ratio between the current policy $\pi_\theta$ and the old policy $\pi_{\theta'}$, the advantage function $A_t$, which describes how much better between two consecutive states $s_{t+1}$, $s_t$, over randomly selecting an action according to $\pi_\theta$, and the policy loss function $L^\pi$ as:

$$L^\pi(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t \right) \right],$$

where $\text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)$ ensures $r_t$ lie in the interval $(1 - \epsilon, 1 + \epsilon)$, and $\epsilon$ is a clip ratio hyper-parameter with typical value 0.2. The value loss $L^V$ is defined as:

$$L^V(\theta) = \mathbb{E}_t \left[ (R_t - V_\phi(s_t))^2 \right],$$

in which $R_t$ represents the traditional $\gamma$-return.

In the HUNL training process, however, the above PPO loss function is difficult to converge. We find two main reasons for this problem: 1) when $\pi_\theta(a_t|s_t) > \pi_{\theta_{old}}(a_t|s_t)$ and the advantage function $A_t < 0$, the policy loss $L^\pi(\theta)$ will introduce a large variance; 2) due to the uncertainty of the opponent’s policy distribution in HUNL (e.g., player performs bluffing), the value loss $L^V(\theta)$ is often too large. To speed up and stabilize the training process of AlphaHoldem, we design a Trinal-Clip PPO loss function. It introduces one more clipping hyper-parameter $\delta_1$ for the policy loss when $A_t < 0$, and two more clipping hyper-parameters $\delta_2$ and $\delta_3$ for the value loss. The policy loss function $L^{t\text{cop}}$ for Trinal-Clip PPO is defined as:

$$L^{t\text{cop}}(\theta) = \mathbb{E}_t \left[ \text{clip}(r_t(\theta), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon), \delta_1) A_t \right],$$

where $\delta_1 > 1 + \epsilon$ indicates the upper bound, and $\epsilon$ is the original clip in PPO. The clipped value loss function $L^{t\text{c}}$ for Trinal-Clip PPO is defined as:

$$L^{t\text{c}}(\theta) = \mathbb{E}_t \left[ \text{clip}(R_t', -\delta_2, \delta_3 - V_\phi(s_t))^2 \right].$$

In training the HUNL AI, the hyper-parameters $\delta_2$ and $\delta_3$ represent the total number of chips the player has placed...
and the opponent has placed, respectively. Thus, these two hyper-parameters do not require manual tuning but are dynamically calculated according to the chips played in the replay. This constriction significantly reduces the variance of the value function, while also eliminates the influence of the policies’ irrationality.

Some previous works also report that clipping on PPO’s policy loss achieves better results in MOBA games (Ye et al. 2020a,b) and MuJoCo (Andrychowicz et al. 2020). Our proposed Trinal-Clip PPO loss further verifies this point in training a HUNL AI. Moreover, this work finds that clipping on the value function further improves the training efficiency and stability significantly, especially for imperfect-information games like HUNL which contains rewarding signals with high variance. The Trinal-Clip PPO loss function improves the learning effectiveness of the actor-critic framework, and we believe it is applicable for a wide range of RL applications with imperfect information.

**Efficient Model Selection and Generation**

With the proposed Trinal-Clip PPO loss function, the most direct way is using the self-play algorithm (Samuel 1959) to train the HUNL agent. However, due to the private information in HUNL, simple self-play learning designed for perfect information game (Heinrich, Lancot, and Silver 2015; Silver et al. 2016, 2018) often causes the agent trapped in a local minimum and defeated by agents with counter-strategies. AlphaStar (Vinyals et al. 2019) designs a population-based training (PBT) procedure to maintain multiple self-play agents and obtains promising results in the real-time strategy game StarCraft II. The PBT procedure needs a tremendous computational resource to ensure good performance.

To obtain a high-performance HUNL AI with both low computation cost and strong decision-making ability, we propose a new type of self-play algorithm which trains only one agent but learns strong and diverse policies. The proposed algorithm maintains a pool of competing agents from the historical versions of the main agent. Then, by competing among different agents, the algorithm selects the $K$ best survivors from their ELO (Vinyals et al. 2019) scores and generates experience replays simultaneously. The main agent learns from the replays and thus can compete with different opponents, maintaining a strong decision-making ability of high-flexible policies. Since the proposed algorithm performs self-play among the main agent and its $K$ best historical versions, we refer to it as $K$-Best Self-Play.

In Figure 4, we compare the proposed $K$-Best Self-Play algorithm against five existing self-play algorithms: 1) the Naive Self-Play (Samuel 1959; Silver et al. 2018), which plays with the agent itself; 2) the Best-Win Self-Play (Silver et al. 2016), which plays with the best agent in history; 3) the Delta-Uniform Self-Play (Bansal et al. 2018), which plays with the agent in the last $\delta$ timestamps; 4) the PBT Self-Play (Vinyals et al. 2019), which trains multiple agents and play with each other; and 5) NFSP (Heinrich and Silver 2016), which plays with the best responses. $K$-Best Self-Play inherits PBT’s merit of diverse policy styles while maintains computational efficiency of single-thread agent training as in Naive, Best-Win, Delta-Uniform self-plays. It also approximates NFSP’s best-response calculation strategy by exploring the policies from the $K$ best agents. As reported in the NFSP paper (Heinrich and Silver 2016), calculating the best response to a HUNL AI from the whole policy space is currently computational prohibitive.

**Experimental Evaluations**

We train the AlphaHoldem model on one computing server with 8 NVIDIA TITAN V GPUs and one AMD 2.00 GHz CPU with 64 cores. For each of the experiments conducted below, including ablations, performance, and comparisons, we use the same quantity of resources to train the model: one ordinary machine with 8 GPUs and 64 CPU cores, unless otherwise stated. AlphaHoldem has a total of 8.6 million parameters, including 1.8 million parameters in the ConvNets and 6.8 million parameters in the fully connected layers.

As for the experimental settings, the mini-batch size per GPU is set to 2,048; thus, the total batch size is 16,384. We use Adam (Kingma and Ba 2015) with initial learning rate 0.0003. For the Trinal-Clip PPO loss, the hyper-parameters $\delta_1$ is set to 3, and $\delta_2$ and $\delta_3$ is dynamically calculated according to the chips played by the players, which range from 0 to 2,0000. The discount factor is set to 0.999. For policy updates, we use GAE (Schulman et al. 2016) with $\lambda = 0.95$ as the advantage estimator. The best performing AlphaHoldem model is trained for a total of 50,000 iterations. During one iteration, there are eight MPI threads, each of which contains 128 environments and 128 steps. Therefore, AlphaHoldem uses a total of 6.5 billion training samples (about 2.7 billion hands). The model winning performance is measured in milli-big-blinds per hand (mbb/h), a standard metric in the poker AI community, representing the average winnings measured in thousandths of the big blinds.

**Ablation Studies**

To analyze the effectiveness of each component in AlphaHoldem, we have conducted extensive ablation studies, as
Table 2: Ablation analyses of AlphaHoldem. Key components include: 1) State representations: Vector, PokerCNN, and W/O History Information; 2) Loss functions: Original PPO Loss and Dual-clip PPO Loss; 3) Self-Play methods: Naive Self-Play, Best-Win Self-Play, Delta-Uniform Self-Play, and PBT Self-Play.

<table>
<thead>
<tr>
<th>Name</th>
<th>Training Time</th>
<th>ELO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td>3.8</td>
<td>78</td>
</tr>
<tr>
<td>PokerCNN</td>
<td>5.4</td>
<td>359</td>
</tr>
<tr>
<td>W/O History Information</td>
<td>6.3</td>
<td>896</td>
</tr>
<tr>
<td>Original PPO Loss</td>
<td>8.4</td>
<td>1257</td>
</tr>
<tr>
<td>Dual-clip PPO Loss</td>
<td>8.4</td>
<td>1308</td>
</tr>
<tr>
<td>Naive Self-Play</td>
<td>8.4</td>
<td>1033</td>
</tr>
<tr>
<td>Best-Win Self-Play</td>
<td>8.4</td>
<td>1024</td>
</tr>
<tr>
<td>Delta-Uniform Self-Play</td>
<td>8.6</td>
<td>931</td>
</tr>
<tr>
<td>PBT Self-Play</td>
<td>8.9</td>
<td>892</td>
</tr>
<tr>
<td>AlphaHoldem</td>
<td>8.4</td>
<td>1597</td>
</tr>
</tbody>
</table>

Figure 5: Loss Curves for Original PPO, Dual-clip PPO and Trinal-Clip among the whole training process. The model with smaller overall loss (shown as blue circles) generally performs better.
cent years, almost all of these AIs are not publicly available, making the comparison between different AIs extremely difficult. To the best of our knowledge, Slumbot (Jackson 2013), the champion of the 2018 annual computer poker competition, is the only publicly available HUNL AI that provides comparisons through an online website2. Slumbot is a strong abstraction-based static agent whose entire policy is precomputed and used as a lookup table. Overall, Slumbot first uses some abstraction algorithms to create a smaller abstract HUNL game. Then it approximates the Nash equilibrium in the abstract game using a CFR algorithm and finally executes the resulting strategy in the original game.

Static AIs like Slumbot suffer from the off-tree action problem, i.e., an action taken by an opponent that is not in the abstraction. A more principled approach is to solve sub-games that immediately follow that off-tree action online. DeepStack and Libratus are representative online AIs based on this idea. We reimplement DeepStack following the original paper’s key ideas and obtain a strong AI named OpenStack3. Specifically, we spend three weeks using 120 GPUs to generate millions of samples to train the value networks. It is worth noting that the creator of Libratus, recently co-authored a paper (Zarick et al. 2020), in which they also reimplemented DeepStack, OpenStack has achieved similar results to theirs, i.e., playing with Slumbot for 100,000 games, OpenStack’s gain is 103.08 mbb/h, which validates the correctness of our reimplementation.

We compare our AlphaHoldem with the above two strong HUNL AIs, i.e., Slumbot and OpenStack for 100,000 hands, and Table 3 shows the head-to-head comparison results. We can see from Table 3 that AlphaHoldem outperforms Slumbot by a large margin. Compared with Slumbot, AlphaHoldem does not require domain knowledge for abstraction and achieves better performance while significantly reducing computational and storage resources. AlphaHoldem also beats OpenStack by 16.91 mbb/h. Unlike OpenStack, AlphaHoldem does not need iterative learning in both the training and inference stages. Given input state representation, it performs only one feedforward pass of the neural network to output the action directly.

To further verify AlphaHoldem’s performance, we evaluate it against four HUNL human professionals. We invite these professional players who have participated in many continental level invitational tournaments, two of whom achieved in the top 10 of continental level tournaments. We ask each player to play against AlphaHoldem for about 2,500 hands. AlphaHoldem beats these professionals by 10.27 mbb/h in average, which supports its high-performance in beating Slumbot and DeepStack.

**Visualization of AlphaHoldem’s Learned Policy**

To analyze AlphaHoldem’s learned policy, we compare the action frequencies where the agent is the first player to act and has no prior state influencing it (Zarick et al. 2020) with those from human professional4, DeepStack, and OpenStack. Figure 6 shows the policies on how to play the first two cards from the professional human and the three agents. AlphaHoldem’s policy is very similar to those of the human professional and the two well-trained agents. These results validate that AlphaHoldem learns a reasonable policy.

**Conclusive Remarks and Future Works**

We have presented AlphaHoldem, an end-to-end reinforcement learning framework to obtain superhuman HUNL AI with the current lowest computation and storage costs and without encoding any human domain knowledge. We achieve this goal through a set of new technical contributions to speed up the training process and simultaneously guarantees the adaptability of the HUNL agent. The proposed learning framework and the speedup training techniques are extendable to the multi-player Texas hold’em and other IIGs like MahJong and Bridge. In future, we plan to expand the proposed framework on more IIG games to promote the development of more general IIG learning frameworks.

---

1https://www.slumbot.com/
2https://holdem.ia.ac.cn/
3https://www.pokersnowie.com/
Acknowledgments

This work was supported in part by the National Key Research and Development Program of China under Grant No. 2020AAA0103401, in part by the Natural Science Foundation of China under Grant No. 62076238 and 61902402, in part by the CCF-Tencent Open Fund, and in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant No.XDA27000000.

References


