AI Writing 5: Final Report

Thomas Knickerbocker
University of Minnesota
Minneapolis, Minnesota, USA
knick073@umn.edu

Ethan Myos
University of Minnesota
Minneapolis, Minnesota, USA
myos0004@umn.edu

Pavle Stojakovic
University of Minnesota
Minneapolis, Minnesota, USA
stoja015@umn.edu

Keywords: Poker, AI, Stochastic, Games

1 Abstract

Texas Hold'Em, a popular and complex variant of poker, has long been a challenging benchmark for artificial intelligence research. The game involves a mix of skill, strategy, and luck, which makes it difficult for AI systems to master. In recent years, AI systems have finally managed to surpass the world's best players, but doing so required significant computational resources, such as supercomputers.

Nonetheless, it is possible to create an AI poker bot that can compete against average human players and operate on a local PC. This bot may not be as advanced as those beating professional poker players, but it still demonstrates an impressive understanding of the game's intricate strategies and tactics.

The architecture of the poker bot, POKI, employs an intricate strategy varying between pre-flop and post-flop stages of a poker round. In the pre-flop stage, it selects moves from a pre-generated table based on the expected value of each move, utilizing Monte Carlo Tree Search to accommodate the vast range of possible outcomes. In the post-flop stage, the strategy involves generating a probability tuple based on the expected hand strength (EHS) and game context. The EHS takes into account both the current strength of the hand and its potential to improve, while the game context adapts based on the opponent's hand and previous decisions.

The bot maintains an opponent model, which includes an action table and a weight table, to predict opponents' actions and adjust its strategy accordingly. This model is updated through a moving weighted average mechanism, ensuring resistance to manipulation. The architecture culminates in the simulator, which runs through multiple scenarios within a set number of trials, combining data from the opponent models and the EHS to make a weighted decision on the bot's optimal next move. This architecture enables the poker bot to execute complex decision-making processes efficiently within a game of poker.

The poker bot was evaluated through self-play tournaments, playing against five randomly selected agents in a 10,000-round game of 5/10 Limit Texas Hold'em. To quantify the bot's performance, the metric of small bets per hand (sb/h) was utilized. The target was an average of > .05 sb/h. The bot surpassed this target with a mean of 0.1032 sb/h, equivalent to an average profit of 50¢ per round, accumulating over $5,000 by the end of 10,000 rounds. The win percentage was 27.6 %, significantly higher than the 18 % expected if wins were purely by chance. Despite these promising results, the high standard error of the mean (0.0282) and the inclusion of values below .05 sb/h in the confidence interval suggest significant volatility and indicate the necessity for further improvements before the bot can compete effectively against human opponents.

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 Introduction

Poker is a classic card game that presents a significant challenge for artificial intelligence (AI) due to its imperfect information structure and partially observable game state, in a multiagent environment. In contrast to games like chess and Go, where perfect information is available to both players, the game of poker involves hidden information and uncertainty, making it more difficult for an AI player to devise a winning strategy[1].

Moreover, a successful poker bot must take into account the opponent's play style and adapt to their strategies, as well as manage risks effectively. This is because an opponent's play style can change throughout the game, and a good player should be able to adjust to these changes to maximize their winnings.

Despite these challenges, recent advances in AI have led to breakthroughs in developing poker-playing programs. For instance, the Pluribus and Libratus programs were developed in 2019 and were able to outperform human players in six-player no-limit Texas hold'em. Prior to these breakthroughs, research on AI players for poker was relatively limited, with most of the literature dating back to the past 25 years [4].

2.1 Texas Hold 'Em

In Texas Hold'em, each player is dealt two cards that are kept hidden from the other players. These cards are known as the player's hole cards. The objective of the game is to make the best five-card poker hand possible using any combination of the player's two hole cards and the five community cards that are dealt face-up in the center of the table. The five community cards are dealt in three stages known as the flop, turn, and river.

Before the first three community cards are shown (also known as the flop), the player to the left of the dealer must place a small blind bet, and the player to their left must place

a big blind bet, which is typically twice the size of the small blind. After the flop, the first round of betting begins, starting with the player to the left of the dealer. Each player has the option to check, bet, or fold. Checking means making no bet, while betting means putting money into the pot. If a player bets, the other players must either match that bet (call) or fold.

After the first round of betting, the fourth community card, known as the turn, is dealt face-up in the center of the table. A second round of betting begins, starting with the player to the left of the dealer. The same options of checking, betting, calling, and folding apply.

Finally, the fifth and final community card, known as the river, is dealt face-up in the center of the table. A third and final round of betting occurs, starting with the player to the left of the dealer. After the final round of betting, if two or more players remain in the game, there is a showdown. The player with the highest ranked hand using any combination of their two hole cards and the five community cards wins the pot.

In the event of a tie, the pot is split evenly among the tied players. The dealer button rotates clockwise after each hand, and the small and big blind bets move around the table accordingly. The game continues until one player has won all of the chips, or until the agreed-upon number of hands have been played.

One of the most important cases for making a bot which is capable of playing and winning poker at a high rate is the very fact that poker is played with money. Money is bet on the outcome of games, and iteratively during rounds of the game, and the game typically ends with one winner keeping all of the money that players entered into the game with for themself. That money is understood to be locked in from the moment the player joins the game, and casinos, among other establishments, host poker tournaments quite frequently, where players may bet money on games. Thus, there is a financial incentive to creating an effective poker bot.

3 Background

3.1 Loki

Early poker bots were constructed using rudimentary rule-based systems alongside hand-crafted decision trees, which offered only limited success against human players [4]. However, in the late 1990s, the University of Alberta's Computer Poker Research Group developed a significant milestone in poker bot technology with the development of Loki. This bot marked a distinct leap forward in poker bot technology as it utilized game-theoretic strategies coupled with hand evaluation techniques [**?**]. Loki's innovative use of information management, which allowed it to maintain and update information about its opponent's potential holdings, also distinguished it from its predecessors [**?**]. These features made Loki a significant stepping stone in poker bot development, and research into poker bot technology began to progress more rapidly.

As researchers delved deeper into more advanced methodologies, abstraction techniques began to emerge. These techniques enabled bots to simplify the complex decision trees involved in poker by grouping similar situations together, reducing the computational complexity associated with decision-making. This approach heralded the development of advanced poker bots like GS1, which combined abstraction techniques with game-theoretic strategies to make more informed decisions [**?**].

3.2 Sparbot, Vexbot

Around this time, the creation of Sparbot and Vexbot marked another significant development in AI poker bot history. Unlike previous bots, Sparbot and Vexbot utilized adaptive strategies and online learning techniques to improve their gameplay during a match. These bots were capable of adjusting their strategies based on the observed actions of their opponents, a crucial aspect of successful poker play. This marked a shift towards a more dynamic and responsive form of poker bot, one capable of learning and adapting to the unique styles and strategies of its opponents [**?**].

Overall, the history of AI poker bots is marked by a series of significant milestones, from the rudimentary rule-based systems of the early days to the more advanced and sophisticated bots of today. Each iteration of poker bot technology has built upon the successes and limitations of its predecessors, leading to the development of increasingly effective and sophisticated bots.

3.3 DeepStack

DeepStack is a revolutionary poker bot that was introduced in 2017 by Moravc ˇ ´ık et al. Unlike other poker bots, DeepStack was designed to be unexploitable by utilizing the concept of Nash equilibrium.

Nash equilibrium is a game theory concept that is used to describe a state in a game where no player can improve their own payoff by changing their strategy, assuming that all other players' strategies remain the same. In other words, it is a state in which no player has any incentive to change their strategy, given the strategies of the other players.

Nash equilibrium is named after John Nash, who was awarded the Nobel Prize in Economics in 1994 for his contributions to game theory. The concept is applicable to many different types of games, including poker, chess, and economics, and has been used extensively in the development of AI algorithms and game-playing bots[8].

In the context of poker, using a Nash equilibrium strategy would be one in which the player's actions are optimized to ensure that they achieve the highest expected value, given the actions of their opponents. This is achieved by computing the optimal strategy for each player, assuming that their

opponents are also playing optimally. By using Nash equilibrium strategies, a poker bot can ensure that it is playing an unexploitable game and that it cannot be beaten by a human player who is also using optimal strategies.
This approach allowed DeepStack to play against human opponents without being taken advantage of, thereby creating a level playing field. The bot was designed as a general-purpose algorithm that could be used for a wide range of sequential imperfect information games, including no-limit Texas hold 'em poker.
To test the effectiveness of DeepStack, the researchers invited 33 professional poker players to play a minimum of 3,000 games against it, for a total of over 44,000 games. The results were amazing. DeepStack's win rate was statistically significant against 10 of the 11 players, demonstrating the power of the bot's approach, as well as its market valuability. This signified a significant milestone in the development of poker bots, as it demonstrated the potential of deep learning and game theory in creating bots that could compete with and even defeat professional human poker players. The success of DeepStack led to a new wave of interest in developing poker bots that could compete at the highest skill and commercial levels of the game [7].

3.4 Pluribus

More recently, the poker bot Pluribus achieved the feat of beating professional poker players in a six-handed game of Texas Hold'em. It learned how to play this specific game by playing five copies of itself. Another recently successful poker bot is Libratus. This bot specialized in heads-up, or two-player, Texas Hold'em. In over 100,000 hands played against four of the best professional poker players, Libratus decisively beat these pros. It used a combination of pre-programmed strategy, adapting to the opponents, and reacting to the gameplay as moves happened each round in order to beat its opponents.

3.5 POKI

POKI is a simulation[5], knowledge-based poker bot developed by the University of Alberta's Computer Poker Research Group (CPRG) in 1999. A pioneer in the field, CPRG's research serves as the foundation for Poker AI; their work has been cited in 100's of subsequent research papers including the one for Pluribus, the Poker AI that conquered the World Series of Poker. [4]. This is considered to be a great accomplishment. Given the relative success of POKI against human play and its reasonably complex architecture, we have determined that it will serve as the main inspiration for our work. POKI consists of four main components: an opponent modeler, hand evaluator, betting strategy generator, and simulator [2].

3.6 POKI - Approach (Basic)

The hand evaluator in POKI is a crucial component of the bot's decision-making process. It determines the strength of the bot's current hand and generates an effective hand strength value that is used to determine the bot's betting actions. The effective hand strength is calculated by considering not only the actual cards in the bot's hand but also the possible hands that the opponent could have based on their previous actions [2].
To calculate the effective hand strength, POKI uses a tree-based algorithm that iteratively constructs and evaluates the possible outcomes of each round of betting. At each step in the algorithm, POKI considers the possible hands that the opponent could have based on their previous actions and updates its estimate of the probability distribution over opponent hands. This allows POKI to make more informed decisions based on the most likely hands that the opponent could have [10].
The effective hand strength is represented as a float value between 0 and 1, where 0 represents a very weak hand and 1 represents a very strong hand. This value is used as input into the betting strategy generator, which determines the bot's betting actions based on a mixed (random) strategy. The mixed strategy generated by the betting strategy generator captures the uncertain nature of poker and allows POKI to be unpredictable, which is an essential characteristic for a successful poker bot [3].
In addition to calculating the effective hand strength, POKI's hand evaluator also takes into account various other factors, such as the position of the bot at the table, the current betting round, and the number of players still in the game. These factors can have a significant impact on the bot's decision-making process, and POKI's hand evaluator is designed to consider all of them [2].

3.7 POKI - Strategy Generator

The betting strategy generator is where the knowledge base of POKI resides. It is a crucial component of the bot as it is called before every betting action, and its output represents the knowledge base of the POKI bot. As cited by [3], the strategy generator outputs a probability triple that represents whether the bot should bet/raise, call/check, or fold. The probability triple generated by the strategy generator represents a mixed or random strategy, as pointed out by [3]. This means that it generates a probability distribution of actions rather than a single action. This approach captures the uncertain nature of poker and enables unpredictability on the bot's part.
It's important to note that the betting strategy generated by the strategy generator is not used directly in action selection, as stated by [1]. Instead, it governs the playout strategy of the simulator and can be thought of as an evaluation function. The betting strategy is used in the calculation of

the expected winning when selecting an action. This means that the betting strategy is an essential element of the bot's decision-making process, and it allows the bot to make informed decisions.
Overall, the strategy generator is a crucial component of the POKI bot, and it plays a significant role in the bot's success in poker games. The mixed or random strategy generated by the strategy generator enables unpredictability on the bot's part, making it harder for opponents to predict the bot's moves. Additionally, the betting strategy generated by the strategy generator allows the bot to make informed decisions by evaluating the potential outcomes of every action.

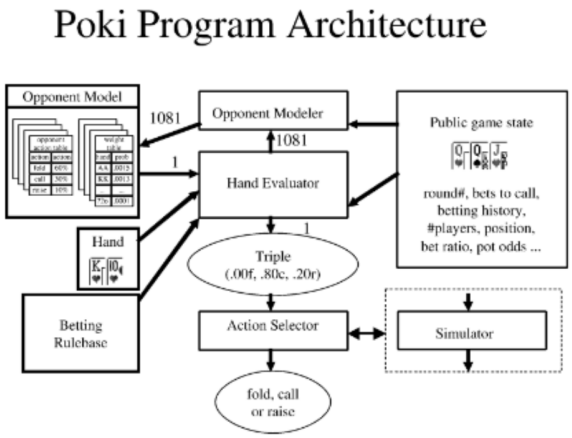
3.8 POKI - Simulator

One of the other major components of POKI is the simulator. As explained by [2], the simulator is called before every betting action and conducts several rollouts of the remainder of the round to inform the bot's decision-making process. This is similar to the Monte Carlo Tree Search (MCTS) algorithm. However, unlike MCTS, the POKI bot's simulator computes the net winnings of each action sequence rather than win probability, as mentioned in [2]. This approach is more advanced as it considers the potential earnings from every action, allowing the bot to make more informed decisions. As pointed out by [1], in each trial, the bot's playout policy is governed by the betting strategy for that particular betting action instance. The betting strategy determines the optimal amount to bet and is a crucial element of the bot's decision-making process.
Moreover, the simulator also exploits selective sampling by assuming the opponent's hand and future actions, using information from the opponent modeler, as mentioned in [1]. This approach allows the bot to evaluate the potential outcomes of every action more accurately.
After conducting 100-500 trials, the bot selects the move that results in the greatest expected winnings, as described by [1]. The expected winnings are calculated by summing the subsequent move's betting strategy and the net winnings from each action. This is a smart way of evaluating the potential outcomes of every action, allowing the bot to make an informed decision.
Therefore, as stated by [1], the betting strategy can be considered the evaluation function during search. The POKI bot uses this function to evaluate the potential outcomes of every action, which allows it to make more informed decisions. Overall, the simulator is a crucial component of the POKI bot, and it plays a significant role in the bot's success in poker games.

3.9 POKI- Opponent Modeling

Opponent modeling is a critical component of POKI's decision-making process, and it serves two primary functions: predicting the opponent's current hand and predicting their future betting actions. The hand evaluator uses the predicted hand information to refine its effective hand strength calculation, while the simulator employs both the predicted hand and predicted moves in a selective sampling approach to reduce computational needs [10]. The opponent modeler in POKI comprises two primary data structures: a weight table and a probability triple [6].
The weight table is used to store the historical frequency of different combinations of board cards and hole cards. This information is used to compute the likelihood of an opponent holding a particular set of cards. The weight table is also used to update the effective hand strength calculation, allowing POKI to adjust its strategy based on its perception of its opponent's hand.
The probability triple is a mixed strategy for predicting the opponent's future betting actions. It represents the likelihood of the opponent betting, checking, or folding. This mixed strategy allows POKI to incorporate randomness into its decision-making, making it more difficult for opponents to predict its actions. The probability triple is calculated based on the opponent's previous actions, including the amount of money they have bet and their position at the table.
By combining information from the weight table and probability triple, POKI is able to generate a more accurate representation of its opponent's playing style and adjust its strategy accordingly. This opponent modeling approach has been shown to be highly effective in improving POKI's performance against human players and other poker bots [6].

4 Methods



4.1 Betting Strategy

The betting strategy of our POKI implementation varies pre-flop and post-flop, which can be summarized as the following: Pre-flop is applied to the period of a game before the first three community cards have been revealed. Here, a move is selected from a table containing each possible hand combination and the expected value for each move. This table is generated pre-game using pure Monte-Carlo Tree Search and

imported into the system, so as to minimize recomputation as games progress. Monte-Carlo Tree Search is particularly useful here since, at this point in the round, there are many many possibilities for both unseen community cards and the opponent's hand, so exploring every possible outcome is far too costly.

Post-flop ranges from when third community card has been revealed to the very end of the round when betting has occurred and the winner has been dealt their earnings. The post-flop strategy consists of taking the expected hand strength, EHS, in conjunction with the game context to generate a probability tuple with three indexes, each representing a different action: betting, checking, and folding. The simulator's usage of this tuple will be covered in a later section.

4.2 Game Context and Expected Hand Strength

The game context is a more static probability tuple that the bot keeps and iteratively adjusts based on the expected value of the opponent's hand and their corresponding decisions from previous rounds. This helps assign biases towards certain decisions (such as checking or raising, in the hypothetical where an opponent has displayed a track record of playing aggressively with relatively low expected hand strength). The calls to these calculations occur during every betting action.

Ahead is the number of hands that the poker bot's cards beat, tied is the number of hands that the poker bot's cards tie, and behind is the number of hands that the poker bot's card loses to. These are all combined in the equation shown in the image below to calculate the current hand strength of the poker bot's hand. The hand strength equates to the probability that a given hand is better than that of an active opponent.

```
HandStrength(ourcards,boardcards)
{
   ahead = tied = behind = 0
   ourrank = Rank(ourcards,boardcards)
   /* Consider all two card combinations of the remaining cards.*/
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank > opprank)         ahead += 1
      else if(ourrank == opprank)    tied += 1
      else /* < */                 behind += 1
   }
   handstrength = (ahead + tied/2) / (ahead + tied + behind)
   return(handstrength)
}
```

The positive potential (PPot) represents the probability that a hand, which is not currently the best, will improve and win at the showdown. On the other hand, the negative potential (NPot) refers to the chance that a hand currently leading will eventually lose. To calculate PPot and NPot, it is necessary to enumerate all possible hole cards for the opponent, similar to the hand strength calculation, as well as consider all potential board cards. Effective hand strength

```
HandPotential(ourcards,boardcards)
{
   /* Hand potential array, each index represents ahead, tied,
      and behind. */
   integer array HP[3][3]   /* initialize to 0 */
   integer array HPTotal[3] /* initialize to 0 */

   ourrank = Rank(ourcards,boardcards)
   /* Consider all two card combinations of the remaining cards
      for the opponent.*/
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank > opprank)      index = ahead
      else if(ourrank = opprank) index = tied
      else /* < */               index = behind
      HPTotal[index] += 1

      /* All possible board cards to come. */
      for each case(turn)
      {
         for each case(river)
         {  /* Final 5-card board */
            board = [boardcards,turn,river]
            ourbest = Rank(ourcards,board)
            oppbest = Rank(oppcards,board)
            if(ourbest > oppbest)       HP[index][ahead ] += 1
            else if(ourbest == oppbest) HP[index][tied  ] += 1
            else /* < */                HP[index][behind] += 1
         }
      }
   }

   /* PPot: were behind but moved ahead. */
   PPot = (HP[behind][ahead] + HP[behind][tied]/2
           + HP[tied][ahead]/2)
          / (HPTotal[behind] + HPTotal[tied]/2)
   /* NPot: were ahead but fell behind. */
   NPot = (HP[ahead][behind] + HP[tied][behind]/2
           + HP[ahead][tied]/2)
          / (HPTotal[ahead] + HPTotal[tied]/2)
   return(PPot,NPot)
}
```

(EHS) is a crucial concept, which combines the current hand strength (HS) and the positive potential. The formula for calculating the EHS for a particular hand is EHSi = HSi + (1 - HSi) × PPoti. This formula takes into account both the current strength of a hand and its potential to improve, providing a more comprehensive evaluation of a hand's overall strength in poker.

4.3 Opponent Modeling

The bot maintains a model for each of the opponents it has in a game. This model consists of an action table and a weight table which is maintained and updated for each opposing player via a Python dictionary where the key to a table is the opposing player's name.

The action table contains the general-case probabilities that some opponent will raise, check, or fold. In our implementation, we initialize the probability table to: [.4, .4, .2] Where the first index represents the chance of the opponent raising, the second checking, and the third folding. These probabilities are adjusted as the opponent plays, based on their actions. A bias of 0.1 is included in each of the table's entries to make the bot more resistant to manipulation by opposing players. Thus, the equation used for updating action table values is a moving weighted average where no value may

drop below 0.1 (and thus, it logically follows that no value may exceed 0.8). The action table is updated by the hand evaluator and opponent history components and is utilized by the simulator as it makes betting decisions.

The second data structure used in opponent modeling, the weight table, is a mapping of all 1326 possible hands to the probability of an opponent holding currently holding that hand. Values are initialized to 1 and set to 0 when the model is sure that the opponent is not currently holding that card. Thus, all entries are initialized to 1 at the beginning of a round (since it is presumed that the deck is shuffled between rounds). Updating the weight table is done by running a probability distribution over the action table and using the product of the weight table entry with This weight table is called by the hand evaluator when calculating potential hand strengths and by the simulator to determine possible hands possessed by an opponent.

```
UpdateWeightTable(Action A, WeightTable WT, GameContext GC,
                  OpponentModel OM)
{
   foreach (entry E in WT)
   {
      ProbabilityDistribution PT[FOLD,CALL,RAISE]

      PT = PredictOpponentAction(OM, E, GC)
      WT[E] = WT[E] * PT[A]
   }
}
```

4.4 Simulation

The simulator is the part of the bot responsible for, essentially, putting it all together. It works by running through a set number of trials, stopping early if someone possesses an obviously preferred move. As stated in previous sections, the opponent's hands are selected by the simulator from the weight table for a specific opponent, within the opponent model. The simulator aims to run as many playouts as possible within the time constraint specified by the parameter MAX_TRIALS. This allows us to put an upper bound on the computational resource consumption of our simulator, and keep decision times for plays within a reasonable bound (anything greater than, say, five seconds will greatly slow down the pace of play, and the consequences of this become radiant when running multiple games).

Thus, the simulator essentially works by running through as many scenarios as possible to get an approximation of how the round may turn out if played to completion, and, in conjunction with the probability tuples gathered from opponent models, makes a weighted decision as to the bot's optimal next action. This can be seen as the command center of the bot.

```
SimulationFramework()
{
   obvious_move = NO
   trials = 0
   while( ( trials <= MAX_TRIALS ) and ( obvious_move == NO ) )
   {
      trials = trials + 1
      position = current_state_of_the_game +
          ( selective_sampling to generate_missing_information )
      for( each legal move m )
      {
         value[m] += PlayOut( position.m, info )
      }
      if( exists i such that value[i] >> value[j] ( forall j ≠ i ) )
      {
         obvious_move = YES
      }
   }
   select move based on value[]
}
```

5 Results

5.1 Challenges in Quantifying Performance

Quantifying the performance of an AI poker bot is challenging due to the inherent complexity and randomness of the game. The outcome of a single hand can be influenced by both the cards dealt and the actions of the players. Moreover, the strategies employed by players can vary widely and are often unpredictable. As a result, even after thousands of hands, the performance of an AI poker bot can still vary significantly from game to game. Additionally, luck plays a significant role in the game, making it difficult to attribute success solely to the skill of the AI.

5.2 Self-Play Tournaments

Self-Play in this context refers to poker simulations where the AI competes against different versions of itself/other bots. In a single replication of self-play, 2,500 hands are dealt[3]. Self-play further reduces variability by mimicking duplicate bridges; the AI plays one round each with all the set of poker cards dealt that hand. In a game with 6 players, this equates to 15,000 rounds. This tournament is then repeated for 500 replications, and the results are aggregated into a single pot. One of the faults occurring with self-play is that it fails to capture the wide variety of play styles, and conditions experienced in a real game. Even with a well-curated, balanced field of bots, the results don't necessarily translate to real playing conditions[2].

5.3 Human Competition

Another option available is testing AI against human opponents. This method is a fan favorite of researchers, early research in the field used internet relay chat (IRC) poker servers to test their programs[6]. Today, many free online lobbies exist where users can test their craft against other human opponents. The human element translates to more reliable results for analysis, however, the method comes with its own challenges as well. One such challenge is the fast-paced nature of human poker, given the sheer amount of

data that needs to be input and processed by the AI. Human poker players are also very adaptable, making them difficult to model in the AI's decision analysis. If one is struggling to find enough opponents for a bot, paying people to play against the bot can be done, which was actually used in the case of DeepStack, for the analysis within their paper [7].

5.4 Selected Approach for Evaluation

The self-play approach was chosen for evaluating the poker bot as it is a widely-used method for testing AI poker bots. In a self-play tournament, the AI plays against itself or different versions of other bots to reduce variability and mimic duplicate bridges. The tournament used a local Python implementation of 5/10 Limit Texas Hold'em, which is a popular variant of poker. The 10,000-round game played by the poker bot agent against 5 random agents was chosen as it provided enough data for meaningful statistical analysis. To ensure the tournament ran to completion, each agent was given an infinite money pot from which to bet. This was done to eliminate the possibility of a bot running out of funds mid-game, which could affect the overall performance of the poker bot. The tournament was hosted on a local Intel i5-12600K system with 32GB of RAM, with a runtime of roughly 1000s (17min). The choice of hardware was likely based on the resources required to run a large-scale tournament and the availability of the hardware to the researchers. /subsectionEvaluating Performance Evaluating AI performance can be done with fixed-limit poker, using a metric known as small bets per hand (sb/h)[9]. Fixed-limit poker is a type of poker game where the bets and raises are limited to a specific, predetermined amount for each round of betting. In most fixed-limit games, there exist two betting sizes, known as the small bet and the big bet. Sb/h measures the net amount of small bets won or lost, divided by the total number of hands played. For example, in a 10/20 game, a 0.1 sb/h is equivalent to an average profit of $1 per hand for the player[9]. When A is greater than or equal to 0.05 sb/h, that indicates a successful AI.

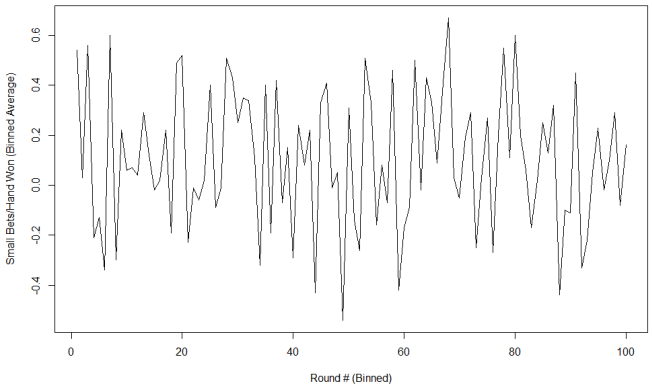
5.5 Data Collection and Analysis

Each round, the number of small bets made by each player as well as the round outcome was recorded in a dictionary object. At termination, these results were exported to CSV and then imported into an R script for statistical analysis. The findings are presented below:

5.6 Findings

The goal was: > .05
Poki had an average of .1032
High variance, as demonstrated by the std error of 0.0282
<.05 is captured in the confidence interval

Figure 1. Time series of small bets won each round for the poker bot. To reduce density and improve the overall legibility of visualization, rounds were binned in 100-round intervals and the mean of each bin was plotted



5.7 Performance Analysis

Overall, the performance objective for the poker bot was an average of > .05 sb/h and the bot surpassed that with a mean of 0.1032 small bets won per hand. This equates to an average profit of 50¢ per round, and indeed the poker bot's cumulative winnings were over $5,000 at the end of round 10,000. The win percentage of the bot was also 27.6%, much better than 18% anticipated if the win percentage was simply due to random chance. However, as anticipated, results were highly volatile, as seen in Figure A. The standard error of the mean was .0282, which is significant given the size. Not only that but values below .05 sb/h were captured. This suggests that further work is required before this bot is ready for human competition.

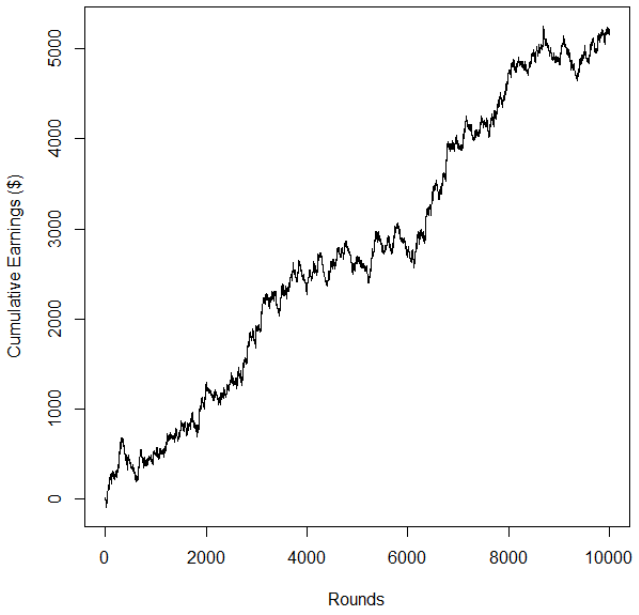
Statistic	Result
sb/h	0.1032
Confidence Interval	[0.0479, 0.1585]
Std Error Sb/h	0.0282
Win Probability	27.57%

Table 1. Summary Statistics of poker bot's overall performance.

6 Conclusion

In conclusion, the poker bot was successfully developed and is capable of playing competitively against average opponents while running on an average PC. The primary challenge was to create a poker bot that could perform competitively against random agents without relying on extensive computing resources often required by more advanced AI

Figure 2. Time Series displaying poker bot's cumulative earnings over the tournament. At over $5000, this equates to roughly 50¢ per round



systems. By focusing on simplified strategies, adaptive learning, and efficient resource utilization, the poker bot demonstrated an impressive understanding of Texas Hold'Em and was able to successfully play against random agents. Further testing with the bot would involve human play.

The success of the project can be attributed to the effective optimization of algorithms, accurate hand evaluation, and opponent modeling. Many different strategies were combined to create a decision-making process that considered many many facets of the game.

A shifting betting strategy allowed for time-efficient computations, and the logical (albeit brute force) approach of the hand strength algorithm allows for accurate and widely-encompassing approximations of hand value. The past decisions of opponents were factored in, and a different model was used for each opponent. Biases were added to models to help minimize the potential for players to exploit the bot's processes, and hand potential was calculated so that the bot could factor in not just what hands currently are, but what they could become in the future. Finally, the bot uses a simulator to put everything together and run through an upper-bounded number of trials, then make a probabilistic decision based on the congregation of all factors. This leads to a bot that (as our results indicate) performs quite well, and is challenging to defeat.

To further improve the poker bot and make additional progress, several steps can be taken. These include continuously refining and updating the bot's strategies based on new insights and emerging trends in poker to ensure that it remains competitive as the game evolves, and enhancing the bot's ability to model a broader range of opponents to adapt more effectively to varying playstyles and strategies.

Incorporating advanced machine learning techniques, such as deep learning and neural networks, could improve the bot's decision-making capabilities and help it identify more subtle patterns in opponents' play. Extending the bot's capabilities to support multi-table tournaments and different poker variants would increase its versatility and broaden its appeal to a wider range of users. Additionally, developing a user-friendly interface and allowing users to customize the bot's strategies and settings would make the system more accessible and appealing to casual poker players and enthusiasts.

By focusing on these areas of improvement, the poker bot can continue to evolve and adapt to the ever-changing landscape of poker, maintaining its competitive edge and offering valuable insights into the development of AI for complex strategy games.

7 Contributions

Ethan contributed to the creation of this paper by composing the introduction, methodology, and analysis sections.

Thomas contributed to the creation of this paper by composing the betting strategy, opponent modeling, and simulator sections, as well as much of the programming implementation of the project.

Pavle contributed to the creation of this paper by composing the abstract, hand strength, and conclusion sections, as well as providing much of the initial research and plan construction for this project.

References

[1] Darse Billings. 2006. Algorithms and assessment in computer poker. (2006).

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

[3] Darse Billings, Jonathan Schaeffer, Duane Szafron, et al. 1999. Using probabilistic knowledge and simulation to play poker. In *In AAAI National Conference*.

[4] Noam Brown and Tuomas Sandholm. 2019. Superhuman AI for multiplayer poker. *Science* 365, 6456 (2019), 885–890. https://doi.org/10.1126/science.aay2400

[5] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 4. 216–217.

[6] Aaron Davidson, Darse Billings, Jonathan Schaeffer, and Duane Szafron. 2000. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI'00*. 1467–1473.

[7] Matej Moravč ík, Martin Schmid, Neil Burch, Viliam Lisý, 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 (may 2017), 508–513. https://doi.org/10.1126/science.aam6960

[8] John F. Nash. 1951. Non-cooperative games. *Annals of Mathematics* 54 (1951), 286–295.

[9] Jonathan Rubin and Ian Watson. 2011. Computer poker: A review. *Artificial intelligence* 175, 5-6 (2011), 958–987.

[10] Gerald Tesauro and Gregory Galperin. 1996. On-line policy improvement using Monte-Carlo search. *Advances in Neural Information Processing Systems* 9 (1996).