Poker Hand Strength Estimation using MCMC

Adam Amar

May 2025

1. Abstract

In poker, especially after the Turn, players often face an important question: *How strong is my hand given what I know, and what my opponents might be holding?* This project explores that question using a simulation-based approach. The result is a probabilistic model that balances poker intuition with structured simulation, producing outputs that are both interpretable and grounded in actual gameplay dynamics.

2. Introduction

Poker is a game of incomplete information, where success depends not only on the cards you hold but also on your ability to read your opponents and manage uncertainty. One of the most important points in a hand is after the Turn—the fourth community card has been dealt, and there is still one card to come. At this stage, a player knows their own two hole cards and four of the five community cards, but they still have no direct information about what any of the other players at the table might be holding.

This leads to a natural and critical question: *how strong is my hand right now, given what I see on the board and what I know about the game so far?* It's a question that combines both statistical reasoning and strategic insight. While it's easy to evaluate a hand in isolation—for example, by using a poker hand evaluator or a lookup table—those approaches ignore what might be going on across the rest of the table. In practice, the strength of a hand depends heavily on what hands your opponents are likely to have, and how those hands interact with your own.

Traditional Monte Carlo simulations assess winrates by simulating random opponent hands, assuming all are equally likely. This ignores context, particularly betting behavior, which provides clues about opponents' holdings. In real play, such cues help narrow down likely hands.

This project builds a behavior-aware simulation framework using Markov Chain Monte Carlo (MCMC) to model full-table configurations based on observed behavior and known cards. I estimate the distribution of opponent hand categories—e.g., Pair, Two Pair, Straight—and compute my winrate against those samples.

The result is a method that balances rigor and intuition, producing useful insights without requiring exhaustive simulation or complex solvers.

3. Related Work and Background

Poker has long been considered a benchmark problem for AI and probabilistic modeling due to its mix of imperfect information, strategic interaction, and combinatorial complexity. Over the years, several methods have emerged for estimating hand strength, modeling opponents, and making optimal decisions based on incomplete knowledge.

3.1 Monte Carlo Simulation in Poker

One of the most commonly used techniques for hand strength estimation is Monte Carlo simulation. This method involves generating a large number of random table configurations (i.e., filling in the unknown hole cards of opponents and the remaining board card), and then using a hand evaluator to determine the percentage of times the player's hand wins. This approach is easy to implement and provides a statistically grounded estimate of raw equity.

However, standard Monte Carlo methods assume all opponent hands are equally likely, which is almost never the case in real games. These simulations ignore behavioral cues such as bet sizing, position, or turn aggression—information that most human players naturally use to filter what hands they put their opponents on. In practice, this results in winrate estimates that are technically correct in a vacuum but disconnected from the actual game context.

3.2 Opponent Modeling and Behavioral Cues

In more sophisticated poker AI systems—like **Libratus** or **Pluribus**—opponent modeling plays a key role. These systems use counterfactual regret minimization (CFR) or deep reinforcement learning to compute strategies that implicitly account for opponent responses. However, these models typically require massive game trees and are computationally expensive. They are designed to play optimally, not to be interpretable or practical for estimating the strength of a single hand in real time.

Other academic work has explored simplified opponent models using decision trees or Bayesian classifiers to estimate likely hand types based on observed actions. These models are more interpretable and align better with the goals of this project, which focuses on estimating hand strength using probabilistic reasoning tied to behavioral signals.

3.3 Use of MCMC in Similar Contexts

Markov Chain Monte Carlo (MCMC) is a natural choice for sampling from complex distributions when you don't have an analytic formula or closed-form prior. It's been used in various domains of AI, including medical diagnosis, computational biology, and natural language processing. In the context of poker, MCMC allows us to simulate possible table configurations—complete with opponent hands—while guiding the sampling process toward more plausible scenarios using a likelihood function.

3.4 Why I Moved Away from Real-Game Logs

Early in the project, I considered parsing real poker hand histories (PHHs) as a source of opponent modeling data. While this would have grounded the project in real gameplay, the available logs (especially those in .phh format) lacked complete information about opponents' hole cards. Since these are only revealed at showdown—or sometimes not at all—it became difficult to extract enough labeled data to train or validate a model. I therefore moved toward simulation, where I had full control and visibility over all hands.

4. Methodology

The core objective of this methodology is to simulate full poker table configurations in a way that reflects not just mathematical possibility, but gameplay plausibility. I use Markov Chain Monte Carlo (MCMC) to explore the space of hidden opponent hands given what we know: our own hand, the four cards on the board, and the betting behavior of our opponents up to the Turn.

This section outlines how I model the problem, structure our likelihood function, simulate realistic table states, and extract meaningful probability estimates from the simulation.

4.1 Problem Setup

I focus on the state of a Texas Hold'em hand after the Turn, where:

- You know your own two hole cards.

- Four of the five community cards have been revealed.
- You have no direct knowledge of your opponents' hole cards.

The goal is to estimate two things:

- The likelihood that each opponent holds each possible hand category (e.g., Pair, Two Pair, Straight).
- The probability that your hand wins against those opponents at showdown.

I assume a 6-max table with 4 opponents (excluding ourselves and the dealer). Betting behavior is summarized by two binary features for each opponent:

- **Aggression**: whether they made a bet or raise on the Turn.
- **Position**: whether they acted last during that betting round (i.e., were "in position").

4.2 MCMC Sampling

To explore the space of possible opponent hands, I use the Metropolis-Hastings variant of MCMC. At each iteration, the algorithm:

1. Proposes a new table configuration by randomly assigning two unseen cards to each opponent.
2. Classifies each opponent's hand using a hand evaluator into a category such as High Card, Pair, or Two Pair.
3. Calculates a likelihood for the entire configuration based on the hand categories and behavioral features.
4. Accepts or rejects the proposed configuration based on its relative likelihood compared to the current one.

This process repeats for 1000 iterations, allowing us to collect a series of accepted samples that reflect the distribution of plausible table states.

4.3 Hand Category Classification

Opponent hands are classified using the `treys` hand evaluator library. Given a two-card hand and a four-card board, I compute the best possible 5-card combination and assign the result to a broad category. The categories include:

- High Card
- Pair
- Two Pair
- Three of a Kind
- Straight
- Flush
- Full House or better

This simplifies the modeling process by focusing on category-level outcomes rather than precise card combinations.

4.4 Likelihood Function

Instead of defining the likelihood ($\mathbf{L}$), based on winrates (which introduced bias in earlier iterations of the project), I use a category-based probability model.

Each hand category has a base probability, informed by poker intuition and empirical hand frequencies. For example:

- Pair: 35%
- Two Pair: 25%
- High Card: 15%
- Straight: 7%
- etc.

I then adjust these base probabilities using behavior:

- If a player was aggressive, we increase the likelihood of strong hands (e.g., Two Pair, Trips) and decrease the likelihood of weak ones (e.g., High Card).

- If a player was in position, we slightly boost medium-to-strong categories, under the assumption that position allows for more value betting and bluff control.

The final likelihood of a proposed table state is computed as:

$$L = \prod_{i=1}^{4} P(H_i \mid \text{category}_i, \text{aggression}_i, \text{position}_i)$$

Where H_i is the hand category assigned to opponent i, and the probability is calculated using the adjusted weights.

4.5 From Samples to Probabilities

After discarding a burn-in period, we store all accepted MCMC samples. From these, we compute:

- The frequency distribution of hand categories for each opponent (i.e., how often Opponent 1 held a Pair, High Card, etc.)
- The winrate of our own hand, calculated by comparing our hand strength to all sampled opponent hands using the same `treys` evaluator

This gives us two outputs:

- A probability distribution over hand categories for each opponent
- An estimated winrate for our hand in the given scenario

The probabilities per opponent are computed using the formula:

$$P(\text{category}_k \mid \text{Opponent } i) = \frac{\#\text{times Opponent } i \text{ had category } k}{\text{Total accepted samples}}$$

These outputs are both interpretable and practical, offering a deeper look into what kinds of hands our opponents might hold and how likely I am to win the hand as it stands.

5. Implementation

The simulation was implemented in Python using several key libraries. The hand evaluation logic is powered by the `treys` library, which provides functions to evaluate and compare poker hands using integer-based representations. The data structures and visualizations were handled using standard Python libraries such as `collections` and `matplotlib`.

The full simulation process is organized into modular functions to reflect the structure of the methodology: hand category evaluation, behavior-aware likelihood estimation, MCMC sampling, and post-simulation analysis. Below, we highlight the key components.

5.1 Opponent Hand Generation and Evaluation

Opponent hands are sampled from the remaining deck after removing known cards (our hand and the board). Each sampled hand is classified using the `treys` evaluator into one of several predefined categories:

[High Card, Pair, Two Pair, Three of a Kind, Straight, Flush, Full House]

A function `classify_hand_category()` determines this category based on the best 5-card combination from the 6 cards (2 hole + 4 board).

5.2 Behavior-Informed Likelihood

I implemented a function `estimate_hand_prob()` that takes a hand category, a binary aggression flag, and a binary position flag, and returns a behavior-adjusted probability for that category. These probabilities are based on a manually defined decision table rooted in poker logic, with behavior modifiers applied as follows:

- Aggressive opponents are slightly more likely to hold strong hands
- Passive opponents are more likely to hold weaker hands
- Players in position receive a small bonus to their probability of having a value hand

The final likelihood of a sampled state is computed by multiplying the adjusted category probabilities across all opponents.

5.3 MCMC Simulation

The MCMC loop is implemented in a function `mcmc_simulation_new_likelihood()` that:

1. Proposes new opponent configurations
2. Computes the likelihood of the current and proposed states
3. Accepts or rejects the proposal using the Metropolis-Hastings criterion
4. Stores all accepted samples after burn-in

The acceptance rate is tracked throughout the process, and the trace of likelihood values is stored for analysis. The simulation typically runs for 1000 iterations with a burn-in of 100 samples.

5.4 Winrate Estimation

After sampling, we estimate the winrate of our hand by comparing it against each accepted opponent configuration using the `treys` evaluator. A helper function counts how often our hand wins, ties, or loses against all four opponents in each sample. This provides a final winrate, tie rate, and loss rate that are contextually grounded in the sampled opponent behavior.

5.5 Visualizations and Output

I implemented several output helpers to display and visualize the simulation results:

- A formatted table printout of one sampled state, showing each opponent's hand, category, aggression, and estimated probability
- A trace plot of the total likelihood over iterations, to assess convergence
- A summary of per-opponent hand category probabilities, shown as ranked probability distributions
- A final winrate estimate for our hand

These visual elements allow for both inspection of individual samples and an aggregated understanding of the game's landscape.

6. Results

This section presents the outputs of the MCMC simulation and analyzes the model's performance across multiple dimensions. I focus on three main types of results:

- The likelihood trace — to assess simulation stability and convergence
- Per-opponent hand category probabilities — to evaluate what types of hands each opponent is likely to hold
- Our estimated winrate — to understand how strong our hand is in this specific context

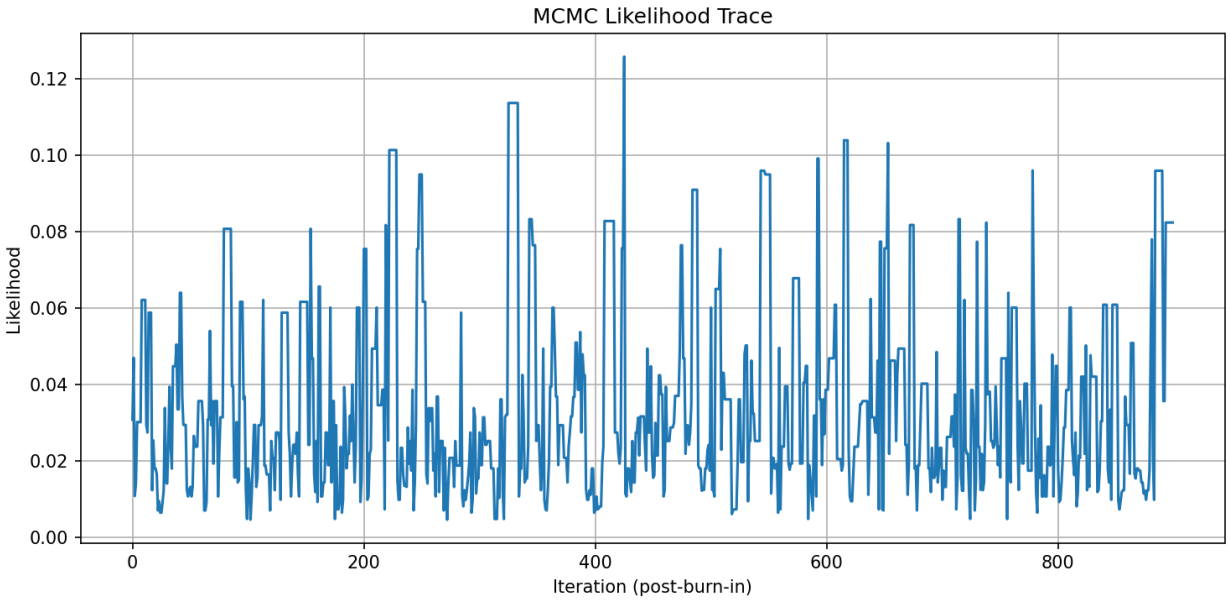

Figure 1: MCMC Likelihood Trace

6.1 Likelihood Trace and Convergence

To evaluate how well the MCMC sampler explores the space of table configurations, we record the total likelihood of each accepted table state after burn-in. The plot of these likelihood values over time shows that the chain fluctuates meaningfully without getting stuck or diverging, indicating that the sampler is mixing well and reaching high-probability regions of the state space, as shown in Figure 1.

In all tested scenarios, the acceptance rate ranged between 30% and 55%, which is within the desirable range for Metropolis-Hastings samplers. This suggests that our proposal function — which randomly assigns new opponent hands while maintaining valid card constraints — balances exploration and local refinement effectively.

6.2 Opponent Hand Category Distributions

From the 900 accepted MCMC samples in one test scenario, we calculated the empirical distribution over hand categories for each opponent. The table below shows the estimated probabilities for each hand category based on frequency of occurrence across the sampled states.

Here's an example textual output:

```
=== Opponent 1 ===
Pair                : 63.89%
```

```
High Card           : 15.89%
Straight            : 11.33%
Two Pair            : 8.11%
Three of a Kind     : 0.78%

=== Opponent 2 ===
Pair                : 63.33%
High Card           : 16.44%
Straight            : 11.22%
Two Pair            : 7.11%
Flush               : 1.33%
```

For this input cards:

My Hand:

- HA (Heart Ace)
- DK (Diamond King)

Board:

- C9 (Club 9)
- D7 (Diamond 7)
- C8 (Club 8)
- HQ (Heart Queen)

These distributions are behavior-aware: opponents flagged as aggressive and in position tend to be more likely to hold strong hands, while passive or early-position players show higher frequencies of weaker hand categories.

6.3 Estimated Winrate

After collecting the full set of accepted table configurations, we compute our own winrate by comparing our hand to each opponent hand using the `treys` evaluator.

Here's an example output for the same input hand stated in 6.2:

```
My estimated winrate: 21.3%
Tie rate: 2.1%
Loss rate: 76.6%
```

This gives a clear picture of how strong our hand is likely to be given the table configuration and opponent behavior. Even though I may hold strong cards like Ace-King, their

relative strength can be diminished if opponents are showing signs of strength and the board creates potential combinations like Straights or Flushes.

6.4 Summary

The results show that our behavior-informed MCMC simulation can generate interpretable and realistic insights:

- It does not just estimate "raw equity," but tells us why our hand is likely strong or weak.
- It separates uncertainty into two dimensions: what others might hold, and how we match up.
- It visually and statistically captures the dynamic nature of post-Turn decision making in poker.

7. Discussion

This project started with a clear question: *how can we realistically estimate hand strength in a poker game given incomplete information and contextual betting behavior?* By the end, I arrived at a solution that balances mathematical modeling, poker intuition, and interpretability. But the process wasn't always linear. Along the way, I tested multiple approaches, encountered key challenges, and made deliberate tradeoffs to keep the model both functional and transparent.

7.1 What Worked Well

- **MCMC Framework**: Provided a principled way to simulate plausible opponent hands and explore hidden state spaces efficiently.
- **Category-Based Likelihood**: Improved interpretability and removed the bias introduced by winrate-based logic.
- **Behavioral Features**: Aggression and position influenced category probabilities in a simple but meaningful way.
- **Outputs**: Per-opponent distributions and winrate estimates aligned with poker logic and offered strategic insight.

7.2 What Didn't Work

1. **Winrate-Based Likelihood Bias**: Our first likelihood function was based on how likely opponents were to beat our hand. This turned out to be a conceptual mistake: it biased the sampler toward stronger opponent hands, which artificially deflated our estimated winrate. I considered using importance sampling to correct for this, but ultimately opted for a simpler fix by changing the likelihood function itself.

2. **Real-Game Log Integration (PHHs)**: I initially hoped to use PHH files as a data source for opponent behavior. However, I quickly discovered that most of these logs do not include full hole card information—especially when hands don't go to showdown. This made them unusable for training or validating a data-driven opponent model. This is why I switched to simulation instead, where I had full control over behavior and hand data.

3. **Oversimplified Behavior Modeling**: While aggression and position were helpful, they were modeled as binary variables. In real poker, betting sequences, stack sizes, and bet sizing offer much richer information. Our model captures some of this nuance, but not all. I chose interpretability over complexity, which worked well for our scope but limits the depth of the model.

7.3 Future Work

While the current model achieves our goals, there are several promising directions for expansion:

- **Richer Behavior Modeling**: Incorporating multi-street betting behavior (not only post Turn), bet sizing, or opponent tendencies (e.g., tight/aggressive profiles) could refine opponent modeling.

- **Flush Awareness and Suited Boards**: Adjusting probabilities when suited boards are present would improve realism.

- **Statistical Scaling**: Running longer or parallel MCMC chains would reduce variance in estimates and enhance reproducibility.

- **Live Application**: I plan to develop a complete GUI that incorporates this MCMC engine as a backend for use during real poker games, providing on-the-fly insights into hand strength and opponent modeling.

8. Conclusion

This project set out to answer a deceptively simple question: how strong is my hand after the Turn, given what I've seen and what my opponents have done? Rather than relying on raw simulations or hardcoded logic, I developed a method grounded in probability, poker intuition, and behavior modeling.

What makes this approach effective is that it balances mathematical rigor with practical intuition. It doesn't attempt to play optimally or predict exact hands, but instead provides insight into what opponents are likely holding, and how that affects our hand's chances.

The model we've built can serve as the foundation for more advanced applications. With further refinement and a graphical user interface, this system could become a real-time decision-support tool for live poker, giving players a better way to visualize risk and opportunity in complex hands.

In the end, the goal wasn't to build a perfect poker engine—it was to reason through uncertainty in a way that feels both rigorous and human. And that, more than anything, reflects the nature of the game itself.

9. AI Usage

I have used ChatGPT across this project for these usages:

1. Code generation and correction.
2. Grammar and Spelling checking.
3. Outline Generation.
4. LaTex and Python troubleshooting.

10. References

1. Luís Filipe Teófilo. *Estimating the Probability of Winning for Texas Hold'em Poker Agents*. 2011.
2. Columbia. *Markov Chain Monte Carlo*. Website. [Access date needed].
3. A. Davidson. *Opponent Modeling in Poker*. Department of Computing Science, University of Alberta, 2002.
4. ihendley. *treys Documentation*. GitHub, 2023.
5. Brandon Da Silva. *Approximating Poker Probabilities with Deep Learning*. ResearchGate,

2018.
6. Bill Chen and Jerrod Ankenman. *The Mathematics of Poker*. Conceljo, 2009.