

Modeling Poker Opponents from Incomplete Hand Histories

Leveraging Bayesian Inference and Neural Networks to Simulate Real-World Stake-Based Behavior

“Real opponents don’t play GTO. Hold’em API models how they actually behave.”

Most poker AI systems are trained using either theoretical models or datasets with complete information—yet in real games, **the majority of hands do not go to showdown**. This introduces a fundamental challenge: **how can we model an opponent’s true range and tendencies when we rarely see their cards?**

This white paper presents the methodology behind **Hold’em API**, a system designed to simulate realistic No-Limit Hold’em opponents using incomplete hand history data, specifically from online micro- to mid-stakes games. The system uses a combination of **Bayesian inference** and **neural networks** to estimate player ranges and predict likely actions—folds, calls, raises, and bet sizes—at various decision points.

Key innovations include:

- A Bayesian framework for **updating opponent ranges** in real time using $P(E | H)$ learned from data
- A neural network model that approximates $P(action, sizing | hand, state, stake\ level)$
- An API interface that makes real player behavior **queryable and pluggable** into bots, simulations, or poker training tools

By modeling players by stake level (e.g., \$0.05/\$0.10 vs. \$1/\$2), Hold’em API captures the **true variation in playstyles across the ecosystem**, making it possible to train agents or build systems that **exploit real tendencies**, not just theoretical ones.

This paper outlines the technical design, data challenges, model architecture, and use cases for developers, researchers, and companies building poker AI or poker-related tools. It bridges the gap between **academic game theory** and the **messy reality of online poker behavior**.

Contents

Introduction	5
Background / Context	6
Our Contribution.....	7
Data Pipeline and Model Inputs.....	8
Hand History Collection	8
Preprocessing Pipeline	8
Input Encoding	9
Model Labels (Supervised Targets)	10
Priors and Assumptions.....	11
Fold vs. Continue Model	11
Objective.....	11
Architecture	12
Training Labels.....	12
Bias Handling.....	12
Example Prediction	13
Raise / Call Model	13
Objective.....	14
Architecture	14
Legal Action Filtering	14

Example Prediction	15
Notes on Labeling.....	15
Integration Flow	15
Bet Sizing Model	16
Objective.....	16
Sizing Buckets.....	16
Architecture	17
Example Prediction	18
Challenges & Adjustments.....	18
Integration Flow.....	19
Bayesian Range Update.....	19
Objective.....	19
Inputs and Outputs	20
Update Examples	20
Implementation.....	21
Use Cases.....	22
Limitations.....	22
Use Cases.....	22
Bot Training.....	23
Game Development	23
Coaching & Leak Detection.....	23
Strategy Research	23
Conclusion	23

Introduction

In No-Limit Texas Hold'em, predicting an opponent's behavior is fundamental to gaining an edge. Yet building reliable models of player behavior is extremely difficult in real-world conditions. Most hand histories contain **incomplete data**, as the majority of hands end before showdown. This means we rarely see what cards a player held—only how they acted.

Despite this, player actions across positions, boards, and stack depths reveal valuable patterns. By modeling these patterns probabilistically, we can infer a player's tendencies and construct **stake-specific behavioral models**. These models are essential for:

- Training AI agents to exploit real player pools
- Simulating lifelike opponents in poker games
- Analyzing decision quality in real-time or post-hand review

This paper introduces the architecture behind **Hold'em API**, a system built to model real poker players using incomplete public hand histories. Unlike traditional GTO solvers or static range estimators, Hold'em API is trained on **millions of actual hands** from micro to mid stakes, and is designed to replicate the behavior of players in those pools as accurately as possible.

Model	Description
Fold / Continue	Given a player's hand, position, stack, and the state of the game, will they fold or continue?
Call / Raise & Check / Bet	If continuing, what type of action will they take? Passive or aggressive? Are they likely to check, call, raise, or bet?
Bet Sizing	If a player chooses to bet or raise, what is the likely sizing bucket (e.g., small bet, half pot, overbet)?

Each stage is handled as a separate but sequential model, allowing us to represent the **conditional decision structure** of real players. Together, these stages allow us to predict **P(action, size | hand, context, stake)** — even when the player's exact hand is unknown.

Throughout the paper, we'll describe:

- How we gather and preprocess real online poker data
- How we model player behavior using neural networks and Bayesian updates
- How the API exposes this intelligence for bot training, simulation, and analytics

By the end, you'll have a detailed view of how realistic, data-driven poker opponent modeling can be achieved—despite imperfect information.

Background / Context

The field of poker AI has seen major advances in recent years, especially in the development of **game-theoretic solvers** like Libratus and Pluribus. These systems have demonstrated superhuman performance in heads-up and multi-player games by computing **Nash equilibrium strategies**. However, they rely on complete information about the game tree, perfect abstractions, and—critically—**do not reflect how actual human players behave** at low to mid stakes.

In practice, most poker hands:

- End **before showdown**, making hole cards invisible
- Involve players who **deviate significantly from optimal strategy**
- Are played with wide **variance in sizing, aggression, and discipline**

As a result, there is a growing need for **pragmatic models** that simulate human behavior—not idealized opponents.

Approach	Description
Static Range Estimators	Tools like Flopzilla and Equilab assume fixed hand ranges per position and don't adjust based on observed behavior.
GTO Solvers and Exploiters	Tools like PioSOLVER compute unexploitable strategies based on tree resolution, not on real player tendencies.
Rule-Based Simulations or Simple Heuristics	Some poker training tools use scripts or logic trees ("If board is dry, bet 30%") which lack depth and adaptability.

None of these systems attempt to **learn from actual online hand histories** with incomplete information. That's where Hold'em API differs.

Our Contribution

We present a system that:

- Trains on **incomplete but abundant data** (millions of online hands)
- Uses **Bayesian inference** to refine hidden opponent ranges
- Predicts real-player behavior using a **three-stage neural model**:
 1. **Fold / Continue**
 2. **Call / Raise & Check / Bet**
 3. **Bet Sizing**

This structure not only reflects how decisions are made in sequence—it also aligns with the observable structure of hand histories, where bet sizes and actions are logged even without knowing the hole cards.

- Different models are created for each stake level, capturing the nuances of how \$0.05/\$0.10 players differ from \$1/\$2 regulars—making it possible to simulate, analyze, and exploit real pools with precision.
-

Data Pipeline and Model Inputs

Modeling opponent behavior from partial information requires careful handling of noisy, biased, and incomplete data. In this section, we describe how Hold'em API collects, processes, and structures data from online hand histories to create training-ready input for all three stages of its behavioral model.

Hand History Collection

We collect publicly accessible **PokerStars hand histories**, recorded either via observer accounts or from parsed logs. These hand histories include:

- Player positions and stack sizes
- Actions (fold, call, raise, check, bet) with timestamps and sizing
- Public board cards per street
- Showdown hole cards (only when revealed)

The result is a massive dataset with **high action visibility** but **partial hand visibility**—ideal for modeling real-world decision points, yet challenging for hand labeling.

Preprocessing Pipeline

Each hand is parsed and split into a series of **decision points**, one per player action. At each point, we extract:

- **Game context:** street, position, stack size, pot size, number of players
- **Board cards:** as revealed so far

- **Prior actions:** sequence of bets/calls/folds prior to decision
- **(If available) Player hand:** only included when revealed at showdown
- **Action taken:** fold, call, bet, raise, check
- **Bet size:** as a fraction of the pot

Each decision becomes a labeled training row for one or more of the three model stages.

Input Encoding

Inputs to the models are encoded as fixed-size numerical vectors:

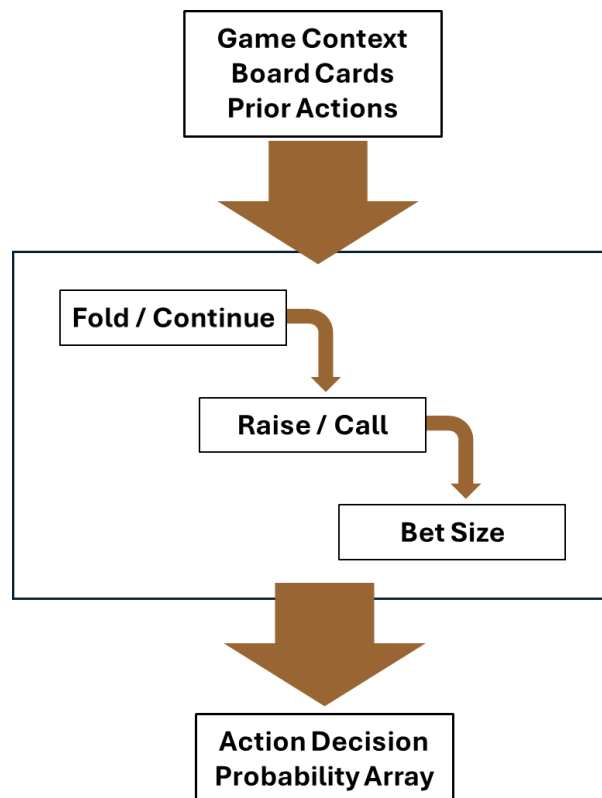
Feature Group	Example Features
Hand Encoding	One-hot vector for combo (if known), masked if unknown
Game State	Position, stack size (in BB), pot size, street
Board Cards	One-hot per suit and rank for each street
Stake Level	Separate models are trained for each stake level to accurately capture differences in player behavior. The stake is encoded as a categorical input or used to route requests to the correct model.

For cases where the player's hand is not known (non-showdown), we treat the hand as a **latent variable** during training and prediction, using priors and updates.

Model Labels (Supervised Targets)

Each model stage has a distinct label set:

Model Stage	Label Type
Fold / Continue	Binary: 0 (fold), 1 (continue)
Call / Raise & Check / Bet	Binary: 0 (call / check), 1 (raise / bet)
Bet Sizing	Categorical: size bucket (e.g., $\leq 25\%$, 25–40%, etc.)



Priors and Assumptions

To handle unknown hole cards:

- We use a **stake-specific preflop range** as the prior (e.g., 22+, A2s+, K9s+ for CO at \$0.10/\$0.25)
- This prior is updated per action using **Bayes' Theorem**, forming a refined posterior range

For training data where hands are revealed, we treat the hand as observed and update model weights accordingly.

This structured pipeline enables us to build robust models from imperfect data—setting the foundation for the **Fold / Continue**, **Action Selection**, and **Sizing Prediction** models described in the next sections.

Fold vs. Continue Model

The first stage of Hold'em API's decision model determines whether a player chooses to **fold** or **continue** at a given decision point. This binary classification is foundational, as it gates all subsequent decisions (e.g., whether a player will raise, call, or bet).

Objective

We model the probability:

$$P(\textit{Continue} \mid H, S)$$

Where:

- H is the player's hand (if known or estimated)
- S is the game state (street, stack size, pot size, position, board, and prior actions)

The model outputs a probability between 0 and 1 representing the likelihood that a player in a specific context will **take any action other than folding**.

Architecture

This model is a binary classifier (logistic regression over a neural net), trained with:

- Input: Encoded vector of hand (when available), position, stack size, pot, board, and prior actions
- Output: Sigmoid activation \rightarrow scalar probability of "continue"

For unseen hands (i.e., non-showdown), we use a prior range distribution $P(H_i)$ and calculate expected continuation probability across all combos:

$$P(\text{Continue} | S) = \sum_i P(H_i | S) \cdot P(\text{Continue} | H, S)$$

Training Labels

Training examples are labeled:

- 0 if the player folds at their decision point
- 1 if the player checks, calls, bets, or raises

Examples with known hole cards are directly supervised. For unknown hands, only the observed action is used during inference or range updates (not for gradient updates).

Bias Handling

Fold decisions are overrepresented in early streets and weak positions. To avoid imbalance:

- We **under-sample folds** in the training set
- Use **stratified batch sampling** to ensure position/street diversity

- Train separate models or embed stake-level priors

Example Prediction

```
Input:
{
  "position": 3,
  "stackSize": 60,
  "potSize": 9.5,
  "board": ["Jh", "8d", "4c"],
  "priorActions": ["open", "call"],
  "street": "flop"
}

Output:
{
  "fold": 0.32,
  "continue": 0.68
}
```

The continuation probability (0.68 in this case) feeds directly into the next stage: **Action Selection** — determining *how* the player continues.

Action Selector Model

Once a player chooses to **continue** rather than fold, the next question is: **what type of action will they take?** Will they call, raise, bet, or check? This is where we model the player's behavioral tendencies conditioned on the decision to continue.

Objective

This model estimates:

$$P(A \mid H, S, Continue)$$

Where:

- $A \in \{check, call, bet, raise\}$
- H is the player's hand (when known or estimated)
- S is the current state of the game (position, stack, pot, board, prior actions, street)

The model outputs a categorical distribution over legal action types.

Architecture

This is a **binary classification model**, trained only on decision points where the player did not fold. It predicts whether the action taken was passive (call/check) or aggressive (raise/bet).

Inputs:

- Encoded hand (when available)
- Position, stack size, pot size
- Board texture and street
- Sequence of prior actions
- Stake level or stake embedding

Output:

- Binary: 0 (call/check), 1 (raise/bet)
-

Legal Action Filtering

The output space depends on the context:

- **Preflop facing a raise:** call, raise
- **Flop first to act:** check, bet
- **Turn facing a bet:** call, raise
- **No bet to call:** check, bet

Illegal actions are masked during prediction and training so that the softmax only distributes probability over valid moves.

Example Prediction

```
Input:
{
  "position": 5,
  "stackSize": 75,
  "potSize": 22,
  "board": ["Qd", "7s", "3s"],
  "priorActions": ["open", "call", "check"],
  "street": "flop"
}

Output:
{
  "check": 0.13,
  "bet": 0.56,
  "raise": 0.00,
  "call": 0.31
}
```

Notes on Labeling

Labels are derived from observed actions **after a fold decision has been ruled out**. When hole cards are known, labels are tied to both the action and hand. When hole cards are not revealed, the action is used to refine the hand distribution (via the range updater, not for gradient updates).

Integration Flow

If a player is predicted to continue (from the Fold/Continue model), the output from this model determines:

Passive vs. aggressive behavior

Whether to proceed to **Bet Sizing** (if action = bet or raise)

Bet Sizing Model

Once a player is predicted to **bet or raise**, the final question is: **how much will they wager?** Real-world opponents use a variety of sizing strategies—some consistent, others erratic—and these vary dramatically by stake level, position, and street.

The Bet Sizing Model estimates the most likely sizing range a player would use **given the context of the hand and their action choice**.

Objective

$$P(S_b \mid H, S, A = \textit{bet or raise})$$

Where:

- S_b is the **bet size bucket** (e.g., 25% pot, 40%, 60%, overbet)
- H is the hand (if known or inferred)
- S is the game state
- The player has already been predicted to **bet** or **raise**

Sizing Buckets

We categorize bet sizes into discrete buckets (based on pot size at time of action):

Bucket ID	Pot % Range	Description
1	≤ 25%	Small bet
2	26–40%	Standard C-bet

Bucket ID	Pot % Range	Description
3	41–60%	Medium sizing
4	61–85%	Large bet
5	86–115%	Pot-size
6	> 115%	Overbet
7	All-in (non-bluff)	Value shove
8	All-in (short stack)	Jam or bust

These buckets are stake-aware—low-stakes players tend to favor certain sizings (e.g., 33%, 100%) much more than others.

Architecture

This model is a multi-class classifier trained on:

- Context features (board, street, pot, position)
- Prior actions
- Player stack and effective stack
- Hand (if known)

It outputs a softmax distribution over the sizing buckets.

Example Prediction

Input:

```
{
  "position": 2,
  "stackSize": 45,
  "potSize": 30,
  "board": ["Td", "9d", "6c"],
  "street": "turn",
  "priorActions": ["open", "call", "check", "bet", "call"]
}
```

Output:

```
{
  "25%": 0.10,
  "40%": 0.51,
  "60%": 0.29,
  "85%": 0.08,
  "Overbet": 0.01,
  "All-in": 0.01
}
```

This output can be sampled stochastically for simulations or used to return the most likely sizing bucket.

Challenges & Adjustments

- **Stack-size capped bets** are normalized so all actions fit within bucketed categories
 - **Bias toward common sizes** (e.g., 33%, 100%) is learned per stake during training
 - **Illegal actions** (e.g., overbet when stack is too short) are masked during prediction
-

Integration Flow

This model is only called if the **Raise/Call Model** predicts a **bet or raise**. The output sizing bucket is passed downstream to simulation engines or bots using Hold'em API.

Bayesian Range Update

In real poker play, we rarely see opponents' hole cards—yet every action they take reveals something about what they might hold. To simulate realistic behavior and refine predictions over time, Hold'em API uses **Bayesian inference** to update a player's hand range based on their observed actions.

Objective

We want to update a player's estimated range $P(H_i)$ after observing an action E (e.g., call, raise, fold). The goal is to compute:

$$P(H_i | E) = \frac{P(E | H_i) \cdot P(H_i)}{\sum_j P(E | H_j) \cdot P(H_j)}$$

Where:

- H_i = a specific hand (e.g., AhKd)
 - $P(H_i)$ = prior probability of that hand being in the player's range
 - $P(E | H_i)$ = likelihood of taking the action E with hand H_i , derived from the **player-decision model**
 - The denominator is a normalization constant over all possible hands
-

Inputs and Outputs

Inputs:

- Prior range (list of hand combos + probabilities)
- Observed action: e.g., "3-bet to 9BB on BTN"
- Game state (street, board, stack sizes, pot, position)
- Stake level

Output:

- Updated hand distribution $P(H_i | E)$, normalized
 - Can be used for next street predictions or equity calculations
-

Update Examples

Example: Preflop 3-bet from Button

```
Prior Range:
{
  "AhKd": 0.022,
  "KJo": 0.016,
  "98s": 0.020,
  "22": 0.015
}

...
...
...
```

```
...
...
...

Observed Action:
{
  "position": 3,
  "action": "3bet",
  "amount": 9,
  "street": "preflop"
}

Updated Range (posterior):
{
  "AhKd": 0.072,
  "KJo": 0.006,
  "98s": 0.003,
  "22": 0.001
}
```

The model increases probability on hands likely to take that action, and sharply reduces weight on speculative or weak hands.

Implementation

This logic powers the /update-range API endpoint. Internally, it:

- Pulls $P(E | H_i)$ from the trained player-decision model
 - Multiplies it with the prior range
 - Normalizes the posterior across all valid hands
-

Use Cases

- **Multi-street simulations** where ranges evolve over time
 - **Bot training:** adversarial bots can adapt based on range narrowing
 - **Coaching tools:** show users what hands an opponent likely held
 - **Equity-aware decision trees** based on updated villain range
-

Limitations

- Assumes player behavior is consistent with modeled $P(E | H_i)$
 - Breaks down if opponent is highly unbalanced or deceptive
 - Garbage priors yield garbage posteriors: initial ranges must be realistic
-

By integrating this update loop, Hold'em API ensures that downstream actions—equity calculations, simulations, and AI predictions—are based not on static assumptions, but on **evolving, realistic opponent behavior**.

Use Cases

Poker software has historically relied on either theoretical models or scripted logic to simulate opponents. But real players—especially in online cash games—don't play by game theory. They play with leaks, patterns, biases, and predictable mistakes.

Hold'em API enables developers, researchers, and product builders to simulate those real behaviors with precision. By querying realistic player actions, bet sizing patterns, and dynamically updated hand ranges, users can build smarter systems that train against or respond to real-world tendencies—filtered by stake, position, and situation.

The following use cases show how the API can be applied across bot development, game simulation, coaching tools, and strategy analysis.

Bot Training

Train poker bots against real-stake players instead of theory. Use `/player-decision` to simulate opponent moves, and `/update-range` to track evolving ranges.

Game Development

Populate poker games with lifelike NPCs that play like actual \$0.25/\$0.50 players using `player-decision` probabilities and bet size realism.

Coaching & Leak Detection

Highlight where a user's decisions diverge from actual pool behavior. Use `range-updates` and `action` likelihoods to illustrate what their opponent probably held.

Strategy Research

Compare EV lines against evolving ranges across streets, or test exploit strategies based on common pool errors at each stake.

Conclusion

Modeling poker opponents in the real world is fundamentally different from solving idealized game trees. Players at micro to mid stakes don't follow GTO—they follow habits, patterns, and pool-wide tendencies. And yet, most available tools either ignore these patterns or lack the data and architecture to model them accurately.

Hold'em API bridges that gap by combining:




- Real-world hand history data (even when incomplete)

- A three-stage neural model of player decisions
- Bayesian range updates to refine hidden information
- Stake-specific behavior profiles across millions of hands

By structuring opponent behavior as a sequential prediction problem—**Fold/Continue** → **Raise/Call/Check/Bet** → **Bet Size**—and exposing that logic through an API, we give developers, poker trainers, and AI researchers access to **practical, realistic, and exploitable models** of human behavior at the table.

Whether you're building bots, simulating games, analyzing hands, or coaching players, Hold'em API offers a new foundation:
not theory—but data.

Learn More

-  Visit: <https://www.holdemapi.com>
-  Docs: <https://www.holdemapi.com/docs>
-  Contact: support@holdemapi.com