Embedding Generative Opponent Models in Reinforcement Learning Agents for Poker

Kai Liu
Cornell University

Xingwei Li
Cornell University

Chenyu Li
Cornell University

January 24, 2026

Abstract

This paper investigates the integration of generative opponent modeling into reinforcement learning (RL) agents for imperfect-information games like poker. We propose a dual-stage framework where an opponent prediction model is embedded within a Proximal Policy Optimization (PPO) agent. Using a simplified Leduc Hold'em environment, we demonstrate that explicit modeling of opponent behavior improves both the learning process and final win rate. Results show a consistent gain in performance with opponent-conditioned inputs, validating the utility of opponent modeling in data-driven strategic adaptation.

1. Introduction

Poker is a classic testbed for strategic decision-making under uncertainty. As an imperfect-information game, it requires players to infer hidden information and adapt to opponents whose strategies are only partially observable. A central challenge in this setting is *opponent modeling*, which plays a key role in effective decision-making for both human players and artificial agents.

Recent poker AIs based on reinforcement learning, such as DeepStack (Heinrich and Silver, 2016) and Libratus (Brown and Sandholm, 2018), have achieved superhuman performance through self-play and equilibrium reasoning. While these methods produce strong and theoretically sound strategies, they are primarily designed to be unexploitable and often fail to adapt to human-like opponents with diverse behavioral patterns.

Generative Artificial Intelligence offers a promising alternative. In poker, generative models can be used to simulate and predict opponents with different playing styles, enabling training and evaluation beyond standard self-play. In this project, we investigate the use of generative models for opponent modeling in poker.

Opponent adaptation is a key challenge in reinforcement learning for imperfect-information games. Traditional RL agents often assume stationary or equilibrium opponents. This work introduces an explicit opponent prediction module—trained as a generative classifier—that is embedded into the PPO agent's state representation. We evaluate whether this integration improves performance in the Leduc Hold'em poker environment.

2. Background Information

2.1. Leduc Rules and Key Concepts

Leduc Hold'em is a simplified variant of Texas Hold'em that retains the core imperfect-information structure of poker while substantially reducing state and action complexity. The game is played with a small deck consisting of two copies of each rank, each player receives a single private card, and exactly one public card is revealed during the hand. Betting proceeds over two rounds—before and after the public card is revealed—with a limited action space and fixed bet sizes. Despite its simplicity, Leduc Hold'em captures key strategic elements such as bluffing, value betting, and belief updating, making it a widely used benchmark for studying reinforcement learning and opponent modeling in poker-like environments.

2.2. Generative Models in Sequential Decision Problems

Generative models provide a principled way to model complex distributions and sequential data. Techniques such as GANs (Goodfellow et al., 2014), diffusion models (Ho and Salimans, 2020), and transformers (Vaswani et al., 2017) have demonstrated strong performance in generating realistic trajectories across a variety of domains.

In reinforcement learning, generative approaches have been extended to imitation learning and policy modeling. Methods such as Generative Adversarial Imitation Learning (Ho and Ermon, 2016) and Decision Transformers (Chen et al., 2021) model entire action sequences, suggesting that generative architectures can capture decision-making behavior over time.

2.3. Generative Opponent Modeling

Generative opponent modeling extends traditional opponent modeling by representing opponents as stochastic generators of behavior rather than fixed or deterministic predictors. Instead of outputting a single most likely action, generative models capture distributions over actions or strategies, allowing agents to reason about behavioral uncertainty, variability, and non-stationarity in opponent play.

In the context of reinforcement learning, generative opponent models can be learned from interaction data and used as predictive components within the agent's decision-making process. Neural network–based opponent predictors, for example, can estimate an opponent's next action or action distribution conditioned on the current state and interaction history. (Ekmekci and S¸irin, 2013) When embedded into the agent's state representation or value estimation, such predictive models enable online adaptation to exploitable or non-equilibrium behaviors, going beyond equilibrium-focused self-play.

3. Research Questions and Approaches

3.1. Research Questions

1. Can neural network-based architectures generate action sequences consistent with game dynamics?

2. How can generated opponents be used to improve RL policy performance?

3.2. Approach

In this work, we consider a modeling direction based on learned generative policies: we build a generative model using multilayer perceptrons (MLPs). The model can be trained synchronically with a reinforcement learning agent to predict opponent actions conditioned on partial game trajectories. Such models offer a complementary, data-driven representation of opponent behavior that does not rely on natural-language conditioning.

Next, generated opponent actions in the previous part are incorporated into reinforcement learning pipelines. By complementing reinforcement training with generated opponent behaviors, we analyze whether it is possible to improve performance compared to the baseline version without opponent modeling.

3.3. Environment and Framework Setup

We study whether explicit opponent modeling can improve reinforcement-learning–based poker agents in a simplified two-player Leduc Hold'em environment. Leduc Hold'em retains the essential imperfect-information and strategic structure of poker while dramatically reducing the state and action space. This simplification avoids the computational burden of full-scale games, enabling faster training and clearer analysis of learned behaviors.

The core idea is to embed opponent behavior prediction directly into the RL agent's decision-making process. We train a generative neural model that predicts the opponent's next action based on the current observable state. These predicted opponent actions are incorporated into the agent's input features, allowing the policy to adapt dynamically based on anticipated opponent behavior.

We implement the environment using the PettingZoo interface, which supports turn-based multi-agent simulation. To isolate the effect of opponent modeling, we adopt a two-stage training framework:

- **Stage 1:** Train a baseline agent using Proximal Policy Optimization (PPO) against a random opponent.
- **Stage 2:** Freeze the Stage 1 policy as a fixed opponent. Train new PPO agents either with or without embedded opponent modeling to compare performance in controlled self-play.

This setup allows us to evaluate the impact of generative opponent modeling on policy learning in a controlled and reproducible environment, highlighting its benefits in terms of adaptability and win rate improvement.

3.4. Reinforcement Learning Agent via PPO

We train the poker agent using Proximal Policy Optimization (PPO), a policy-gradient method designed for stable on-policy reinforcement learning. PPO is well suited to our online poker setting, where trajectories are generated sequentially through interaction with the environment and the opponent.

State and Action Representation. The agent operates in a two-player Leduc Hold'em environment with a compact but expressive observation space. Each observation is a 36-dimensional vector encoding private cards, public cards, and chip counts for both players. Actions are discrete and correspond to the standard Leduc betting options: `call`, `raise`, `fold`, and `check`.

Table 1: Leduc Hold'em State Representation and Environment Specifications

Index Range	**Description**
0–2	Current player's private card (J, Q, K)
3–5	Public (community) card (J, Q, K)
6–20	Current player's chip count (one-hot encoded)
21–35	Opponent's chip count (one-hot encoded)

Property	**Value**
Number of agents	2
Action space	Discrete (4)
Observation shape	$(36,)$
Observation values	$[0, 1]$

Policy and Value Networks. The PPO agent consists of two neural networks:

- A policy network (actor) that maps the current state to action logits over the four legal actions.
- A value network (critic) that estimates the state value $V(s)$, which is used for advantage estimation.

Both networks are implemented as multilayer perceptrons (MLPs) and are trained jointly during policy optimization.

Advantage Estimation. To reduce variance while maintaining low bias, we use Generalized Advantage Estimation (GAE) with TD(λ). Given rewards r_t and value estimates $V(s_t)$, the advantage is computed as

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t),$$

where γ is the discount factor and λ controls the bias–variance tradeoff.

PPO Objective. PPO updates the policy by maximizing a clipped surrogate objective:

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t \Big) \right],$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)},$$

and ϵ is the clipping parameter that constrains policy updates.

Key Properties. This PPO-based design provides several advantages for our poker setting:

- Stable policy updates through clipped optimization.
- On-policy learning, which aligns naturally with online self-play and opponent interaction.
- Compatibility with opponent modeling, as state representations can be augmented with predicted opponent behavior without changing the optimization framework.

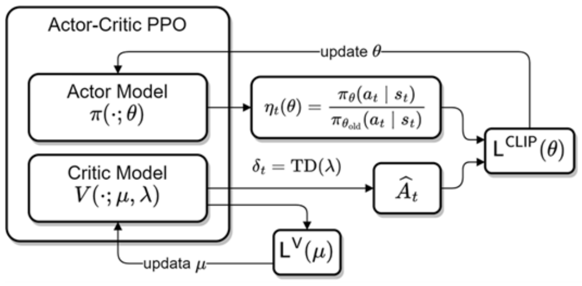
Actor-Critic PPO
update θ
Actor Model
$\pi(\cdot;\theta)$
$\eta_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$
$\mathsf{L}^{\mathsf{CLIP}}(\theta)$
$\delta_t = \text{TD}(\lambda)$
Critic Model
$V(\cdot;\mu,\lambda)$
$\widehat{A}_t$
$\mathsf{L}^{\mathsf{V}}(\mu)$
updata μ

Figure 1: PPO architecture

3.5. Opponent Modeling via MLP

To enhance policy adaptation, we incorporate an explicit opponent prediction model that estimates the opponent's next action from the current state. Unlike persona-conditioned LLMs used in Part 1, this model is fully data-driven and integrated into the reinforcement learning framework.

Opponent Predictor Network. The opponent model is implemented as a lightweight multilayer perceptron (MLP) classifier. It receives the same 36-dimensional observation vector as the agent, which encodes the current hand, public card, and chip counts. The model outputs a probability distribution over the opponent's four possible actions: `call`, `raise`, `fold`, and `check`.

Architecture Details. The network has two hidden layers with ReLU activations, each with 32 neurons, followed by a softmax output layer. The architecture is simple but effective enough in capturing short-term behavioral signals.

- **Input:** Current 36-dimensional observation
- **Hidden Layers:** Two fully connected layers with ReLU (each 32 units)
- **Output:** Softmax over 4 opponent action classes

Training is performed online during PPO agent learning using cross-entropy loss and the Adam optimizer. Each time the opponent acts, the corresponding state-action pair is added as a supervised learning example for the model.

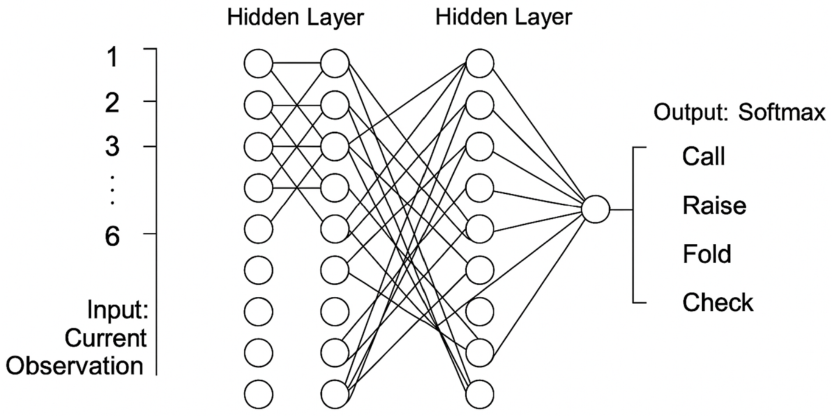

Figure 2: Opponent predictor neural network architecture.

Integration into RL Agent. To incorporate predicted opponent behavior into decision-making, the PPO agent augments its input state with the predicted opponent action probabilities. Specifically, the 4-dimensional output of the opponent model is concatenated with the raw observation vector, yielding a 40-dimensional input to the policy and value networks.

This setup enables the PPO policy to condition its actions not just on the observed state, but also on a learned model of the opponent's likely behavior. The joint training setup reflects a live gameplay environment, where both the policy and its internal belief about the opponent evolve together.

This integrated architecture allows the agent to dynamically adapt to different opponent tendencies and offers a foundation for more behavior-aware reinforcement learning in imperfect-information games like poker.

3.6. Training Settings

We design a two-stage training framework to isolate the effect of explicit opponent modeling within a reinforcement learning pipeline and enable controlled comparisons.

Stage 1: Baseline Training. In the first stage, a PPO agent is trained against an opponent that follows a uniformly random strategy. This setting exposes the agent to a wide variety of game situations while avoiding strong strategic bias from the opponent. The objective of this stage is to learn a reasonable but naive baseline poker policy that captures fundamental betting and folding behavior. After convergence, the trained PPO policy is frozen and used as a fixed opponent in subsequent experiments.

Stage 2: Evaluation and Comparison. In the second stage, we evaluate whether opponent modeling improves learning and performance. The frozen PPO agent obtained from Stage 1 serves as a fixed opponent, ensuring a stationary and controlled evaluation environment. Two agents are trained and compared under identical conditions:

- A PPO agent without opponent modeling, which observes only the raw environment state.

- A PPO agent with opponent modeling, which receives an augmented state that includes predicted opponent action probabilities.

This setup allows performance differences to be attributed directly to the inclusion of opponent modeling.

Opponent Model Training. The opponent predictor is trained online and synchronously during gameplay. At each opponent decision step, the observed state and the opponent's executed action are treated as a supervised learning example. One gradient update is performed per opponent action, mimicking a live, non-stationary gameplay scenario in which the agent continuously refines its beliefs about opponent behavior as new evidence becomes available.

Key Hyperparameters. Across all experiments, we use a consistent set of hyperparameters to ensure comparability:

- PPO training iterations: 30
- Episodes per iteration: 1000
- Advantage estimation: TD(λ) with $\lambda = 0.9$
- PPO clipping parameter: $\epsilon = 0.2$
- Optimizer: Adam for both policy and value networks

These settings balance training stability and computational efficiency while remaining suitable for the simplified Leduc Hold'em environment.

3.7. Performance Evaluation

We evaluate the performance of PPO agents with and without opponent modeling across two training stages, and analyze the quality of the opponent model itself.

Stage 1: Baseline PPO Training

In the first stage, the PPO agent is trained against a random opponent. As shown in Figure 3, the agent's average reward increases steadily and the win rate quickly rises to around 90%. This indicates successful convergence to a competent baseline policy.

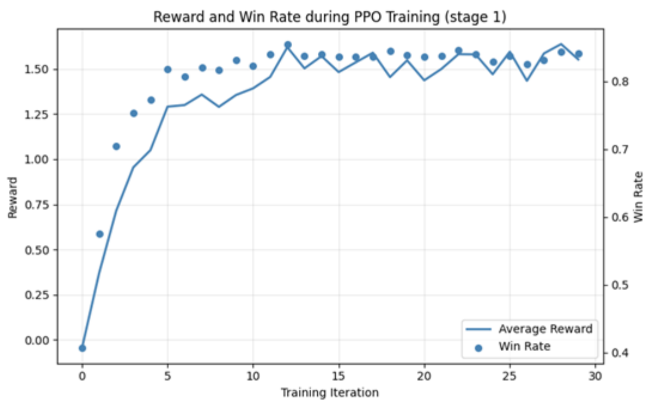

Figure 3: Reward and win rate during PPO training against random opponent (Stage 1).

Opponent Model Learning Quality

The opponent predictor network learns rapidly during training. As shown in Figure 4, the training loss decreases sharply, while prediction accuracy reaches and maintains around 90% after approximate 2,000 steps. This confirms that opponent behavior is learnable from raw observations, even under online, non-stationary conditions.

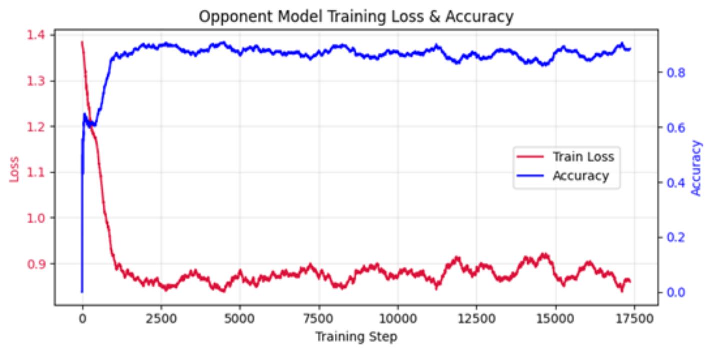

Figure 4: Opponent model training loss and accuracy over time.

Stage 2: Agent Comparison with vs. without Opponent Modeling

In the second stage, two PPO agents are trained against the fixed policy from Stage 1. As shown in Figure 5, both agents show improved reward and win rate over time.

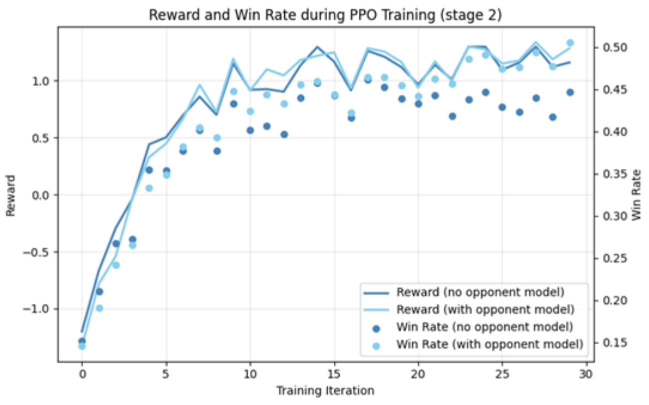

Figure 5: Comparison of reward and win rate between agents with and without opponent model (Stage 2).

Final Metrics

At the end of training:

- Final win rate (no opponent model): 44.7%
- Final win rate (with opponent model): 50.6%
- Improvement over baseline: +5.9 percentage points (≈13.2% relative gain)

These results validate the effectiveness of embedding generative opponent models in reinforcement learning agents. Even a simple predictive model provides actionable behavior signals that improve policy adaptation and overall performance in imperfect-information games.

4. Future Works and Potential Improvements

Several enhancements can be made to the reinforcement learning framework with embedded opponent prediction. More nuanced training settings—such as experience replay, curriculum learning over opponent skill levels, and asynchronous or delayed updates—can improve training stability and mitigate feedback loops between the policy and the opponent model.

Moreover, richer representations of opponent behavior could be achieved by modeling long-term patterns such as bluff frequency, aggression sequences, or street-dependent tendencies. Recurrent architectures or trajectory-level embeddings may provide a more expressive and temporally coherent understanding of opponent styles.

Finally, adapting this framework to more complex poker variants—including multi-player Leduc or full-scale Texas Hold'em—requires additional components such as public belief modeling and deeper action abstraction layers. These improvements would help handle the increased uncertainty and strategic diversity in larger, more realistic game settings.

5. Conclusion

This study shows that generative opponent modeling improves the performance of RL agents in poker. Embedding an action predictor into PPO policies enables agents to better anticipate and exploit opponent behaviors, supporting more adaptive and robust decision-making in imperfect-information settings.

References

Brown, N. and Sandholm, T. (2018). Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*.

Chen, L., Lu, K., Rajeswaran, A., and et al. (2021). Decision transformer: Reinforcement learning via sequence modeling. *NeurIPS*.

Ekmekci, O. and S¸irin, V. (2013). Learning strategies for opponent modeling in poker. In *Computer Poker and Imperfect Information: Papers from the AAAI 2013 Workshop*, pages 6–12.

Goodfellow, I. et al. (2014). Generative adversarial nets. *NeurIPS*.

Heinrich, J. and Silver, D. (2016). Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*.

Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *NeurIPS*.

Ho, J. and Salimans, T. (2020). Denoising diffusion probabilistic models. *NeurIPS*.

Vaswani, A. et al. (2017). Attention is all you need. In *NeurIPS*.