More Than Win or Loss: An ML-IRL Approach for Learning Dense Reward Functions in Chess

Ever Miller Wilson Sun

{evermiller,wilsonsun}@arizona.edu

Abstract

Reinforcement learning (RL) systems for chess, exemplified by AlphaZero, achieve superhuman performance despite relying almost exclusively on sparse terminal rewards—signals observed only at the conclusion of the game. Such sparsity exacerbates the temporal credit assignment problem and demands extensive exploration, as the contribution of intermediate moves to eventual outcomes remains opaque. This work addresses these limitations by introducing am inverse reinforcement learning (ML-IRL) framework that infers dense, intermediate reward signals directly from expert demonstrations. We construct a custom chess environment following the Gymnasium API and develop a residual convolutional reward model that maps temporally stacked board states to bounded scalar rewards reflecting expert evaluative preferences. Using an adversarial IRL training loop, the reward model is iteratively refined to distinguish expert trajectories from those generated by a Maskable-PPO policy network. The learned reward function is then used to train RL agents capable of leveraging continuous feedback rather than terminal outcomes alone. Through experiments using grandmaster games, World Chess Champion games, and engine-generated data, we evaluate the stability and generalization of several IRL reward architectures and analyze their impact on downstream policy learning. Our results demonstrate that dense reward models learned via ML-IRL can capture expert-like positional assessments and enable more sample-efficient training than sparse-reward baselines, providing a promising pathway toward interpretable and data-efficient RL in complex strategic environments.

1 Introduction

1.1 Motivation

Chess has long been regarded as a central benchmark for evaluating strategic decision-making algorithms. Modern reinforcement learning (RL) agents such as AlphaZero have reached superhuman performance, yet they rely almost entirely on sparse reward signals—typically +1 for a win, -1 for a loss, and 0 for a draw, all received only at the conclusion of the game. This sparse reward structure introduces the temporal credit assignment problem: when a game spans 40–60 moves, it becomes difficult to determine which earlier actions were responsible for the eventual outcome. A win may reflect a well-calculated sacrifice on move 12, an opponent's blunder on move 40, or subtle accumulative advantages. Learning exclusively from terminal outcomes therefore requires enormous amounts of exploration and computational resources.

Dense rewards—intermediate feedback after each move—can accelerate learning dramatically. However, constructing a human-authored dense reward function for chess is effectively impossible. Although players can assign coarse values to pieces, the true value of a position is highly contextual. A material advantage may be losing in the presence of tactical threats; conversely, sacrifices may

Preprint. Under review.

be winning due to long-term strategic compensation. Hand-crafted heuristics fail to capture these interactions without invoking search procedures that RL seeks to replace.

This project aims to bridge this gap by automatically generating dense reward signals from expert demonstrations. Using machine-learning-based inverse reinforcement learning (ML-IRL), we infer the implicit reward structure that underlies expert decision-making, enabling the RL agent to receive continuous evaluative feedback without relying on hand-engineered heuristics.

1.2 Related Work

Research on high-level chess decision-making traditionally falls into two paradigms: heuristic search and deep reinforcement learning.

Heuristic Search and Hand-Crafted Evaluation Classical engines such as Deep Blue and early Stockfish versions employ minimax search with alpha–beta pruning. Their performance depends critically on a hand-crafted evaluation function that linearly combines features such as material balance, mobility, and king safety. However, as noted by Shannon (1950) and Turing (1953), these methods suffer from the "knowledge bottleneck": human-specified evaluation terms cannot fully capture the strategic complexity of chess.

Deep Reinforcement Learning with Sparse Rewards The emergence of deep RL shifted the paradigm. Silver et al. (2017) demonstrated that AlphaZero can learn from scratch using self-play combined with Monte Carlo Tree Search (MCTS). Although successful, this approach depends entirely on sparse terminal rewards, leading to high sample complexity and slow convergence due to the difficulty of attributing long-term outcomes to intermediate actions.

Inverse Reinforcement Learning (IRL) IRL, first formalized by Ng and Russell (2000), reverses the classical RL problem by attempting to infer a reward function that explains expert behavior. Maximum Entropy IRL (Ziebart et al., 2008) provides a probabilistic formulation that remains robust even with suboptimal demonstrations. In the context of complex games, ML-IRL enables the automatic extraction of a dense reward function from human or engine games, approximating grandmaster evaluation without relying on manually designed heuristics.

1.3 Objectives

The goal of this project is to develop and evaluate a machine-learning-based IRL framework capable of learning dense reward signals from expert chess games. Our objectives are threefold:

Algorithm Development We design a Maximum Entropy Deep IRL architecture that maps tensor-encoded chess board states to scalar reward values.

Dense Reward Extraction We train the model on expert trajectories to produce dense rewards correlated with move quality, effectively capturing expert intuition in numerical form.

Policy Optimization and Validation We use the learned reward function to train a new RL agent and compare its convergence behavior and performance stability against a baseline agent trained solely on sparse terminal outcomes.

2 Environment Setup

To support RL training and IRL trajectory sampling, we developed a custom chess environment following the Gymnasium API. The environment integrates python-chess for rules and PyTorch for neural evaluation.

2.1 State Space

The symbolic nature of chess requires transformation into a machine-interpretable tensor representation. Each board position is encoded as a tensor of shape (16, 8, 8), where twelve planes represent the

presence of each piece type for both colors, and additional planes encode metadata such as castling rights or repetition indicators. To maintain consistency across roles, all observations are presented from the agent's viewpoint: if playing Black, the board is rotated 180 degrees and piece colors inverted.

Because chess contains history-dependent rules, we adopt temporal stacking. Using a history length of k = 4, the final observation is a tensor of shape (64, 8, 8), enabling the agent to capture temporal patterns relevant to evaluating positions.

2.2 Action Space

Moves are mapped to a discrete action space of size 4,672, representing all theoretically possible from-square/to-square combinations—including promotions—indexed through an ActionIndex encoder. Since only a subset of moves is legal in any given position, the environment provides an action mask for legal moves. A wrapper ensures that illegal selections are penalized and replaced with valid moves, avoiding premature episode termination.

2.3 Reward Structure

2.3.1 Sparse Rewards

The baseline reward function provides feedback only at game termination: +1 for victory, -1 for defeat, and 0 for draws or intermediate states. This mirrors the sparse reward structure used in self-play systems such as AlphaZero.

2.3.2 Dense Rewards via IRL

To address sparse reward limitations, we introduce a learned reward function $R_\theta(s)$. The RewardEnv wrapper computes rewards by forwarding each state through a neural reward model, enabling step-wise feedback.

2.3.3 Network Architecture

The reward model employs a residual convolutional neural network (ResNet). The 64-channel input tensor passes through an initial 3×3 convolution, batch normalization, and ReLU activation. A residual block, consisting of two convolutional layers with a skip connection, extracts high-level spatial features. A final convolution reduces the feature map to 16 channels; after flattening, two fully connected layers map the vector to a scalar output.

2.3.4 Activation and Bounding

To ensure numerical stability, the model outputs

$$r = tanh(Linear(h))$$

bounding rewards within (-1, 1). This prevents divergent Q-value accumulation during RL training.

2.3.5 Integration and Opponent Modeling

At each environment step, RewardEnv replaces the standard reward with the predicted dense reward. The environment supports self-play with configurable opponents, allowing curriculum learning with progressively stronger adversaries.

3 Methodology

This study adopts an adversarial IRL framework that jointly learns a reward function from expert data and trains an RL agent to maximize the learned reward.

3.1 System Architecture

The system consists of three modules: the simulation environment, the neural network models, and the alternating training loop.

Simulation Environment Built on python-chess and the Gymnasium interface, the environment provides temporally stacked observations and action masking. The RewardEnv wrapper injects dense rewards in place of sparse terminal outcomes.

Neural Network Models Both the policy network and the reward model share a residual CNN backbone designed to process the 64-channel input tensor. For policy network, the CNN extracts spatial features, followed by a linear projection into a 512-dimensional vector. Training uses Maskable Proximal Policy Optimization (Maskable-PPO), which ensures probability mass is distributed only across legal moves. The reward model processes the same stacked observation input and outputs a scalar reward through a Tanh-activated final layer, representing the "expert-likeness" of a position.

3.2 Experimenntal Setup

Training proceeds through three phases, orchestrated by *irl_chess.py*.

Phase 1: Expert Data Loading and Pre-training Expert trajectories are loaded and frame-stacked to match the agent's observation structure. Prior to IRL training, the policy undergoes supervised imitation learning for 10 epochs to mitigate the cold-start problem.

Phase 2: Adversarial IRL Loop. Training alternates for a select number of iterations between policy updates and reward updates:

Phase 3: Self-Play Curriculum To ensure robustness, the agent is evaluated every three iterations against an opponent pool. When the agent surpasses a 55% win rate, its policy is frozen and added to the pool, which maintains a fixed capacity of five models. This curriculum ensures continual progression against increasingly challenging opponents.

4 Experimental Results

4.1 Models

We evaluated the training stability by tracking the IRL reward loss as well as the different metrics of the PPO agent. We ended up with four final trained models.

Model 1 was trained with a higher number of ResNet blocks and ended up vastly overfitting the expert data, assigning a reward of -1 to all states not found in the expert dataset. This model was trained on a selection of 2500 high-elo chess games and was trained for approximately 2.5 million time steps with 250 IRL iterations.

Model 2 was trained with a lower number of ResNet block, we also applied a higher entropy coefficient to our PPO model to encourage higher exploration, we also increased the weight penalty of the reward model and decreased learning rate.

Model 3 was trained on a selection of 40 games from the World Chess Champion, Magnus Carlson. This model was only trained for 50 iterations of the IRL algorithm, and we used a significantly decreased model size.

Model 4 used the same configurations as model 3 but was trained on games from the best chess engine, Stockfish.

4.2 Trajectory Analysis: Overfitting

A critical challenge in Adversarial IRL is preventing the reward model from becoming a "perfect discriminator" that simply memorizes the expert dataset. We observed two distinct training behaviors.

We fed a Grandmaster game into the trained Reward Model to see if it could track the flow of the game. These graphs indicate memorization and overfitting, where the model resorts to assigning a reward of one to all states for the white player.

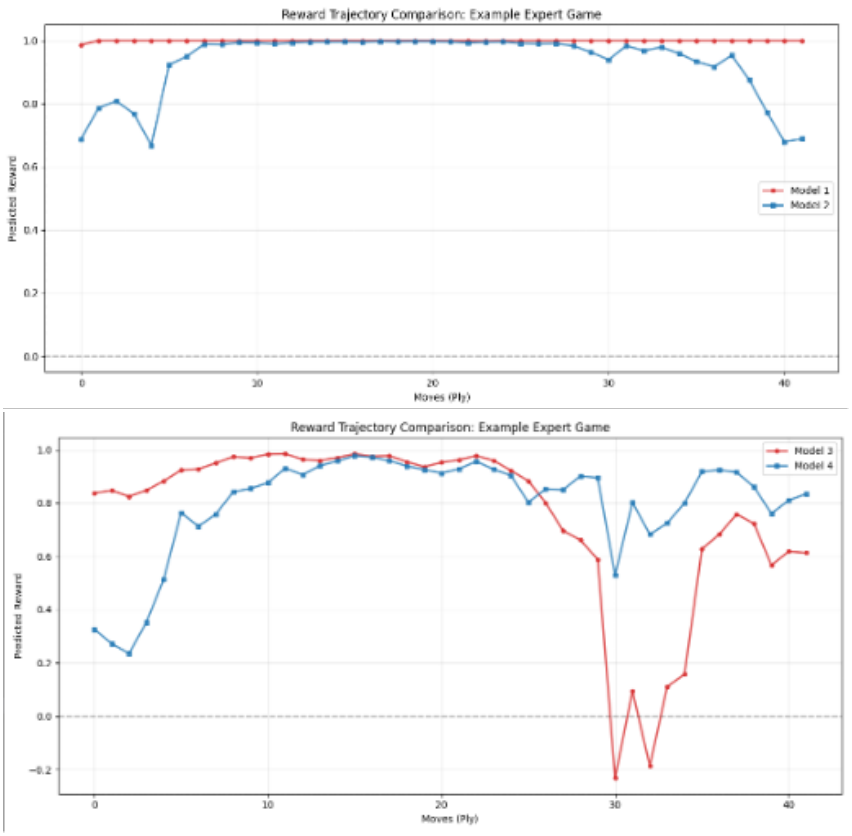

Figure 1: Reward Trajectory Comparison

These trajectories show a clear difference and perhaps overfitting the more trained models, with the most extreme being Model 1.

4.3 Saliency Maps: Visualizing the Reward

By computing the gradient of the predicted reward with respect to the current state, we visualize exactly which squares the model is “looking at” to judge a position.

Model 1 predicts a near-perfect reward but also shows scattered noise in its saliency. Model 2 however, with higher entropy and a less complex model, predicts a more conservative reward. Its map is more active (red areas) and distributed across its pawn structure. This highlights the importance of regularization in IRL tasks. Model 1’s high confidence (0.98) in the starting position is symptomatic of overfitting; it recognizes the exact setup as "Expert." Model 2’s lower confidence (0.68) is more realistic; the starting position is neutral, not winning. The distributed saliency in Model 2 indicates it is attended to broad structural features (the pawn rank) rather than specific pixel values. Model 3 (Magnus Carlson) assigns a high reward to the opening state. The saliency focuses on specific pieces (Knights, Bishops and Queens), reflecting a human-centric opening theory. Model 4 (Stockfish) assigns a significantly lower reward. The saliency is concentrated more on the center file of the enemy side.

The disparity in reward confidence is interesting. The model easily learned to classify Magnus’s moves as good, likely because human play relies on visual pattern recognition, which CNNs are great at imitating. However, Stockfish’s good moves rely on deep position searching and complex heuristics that have no immediate visual indicator. It seems the IRL Model struggled to find a visual feature map that correlated with Stockfish’s evaluation, resulting in a low reward confidence even for the opening position.

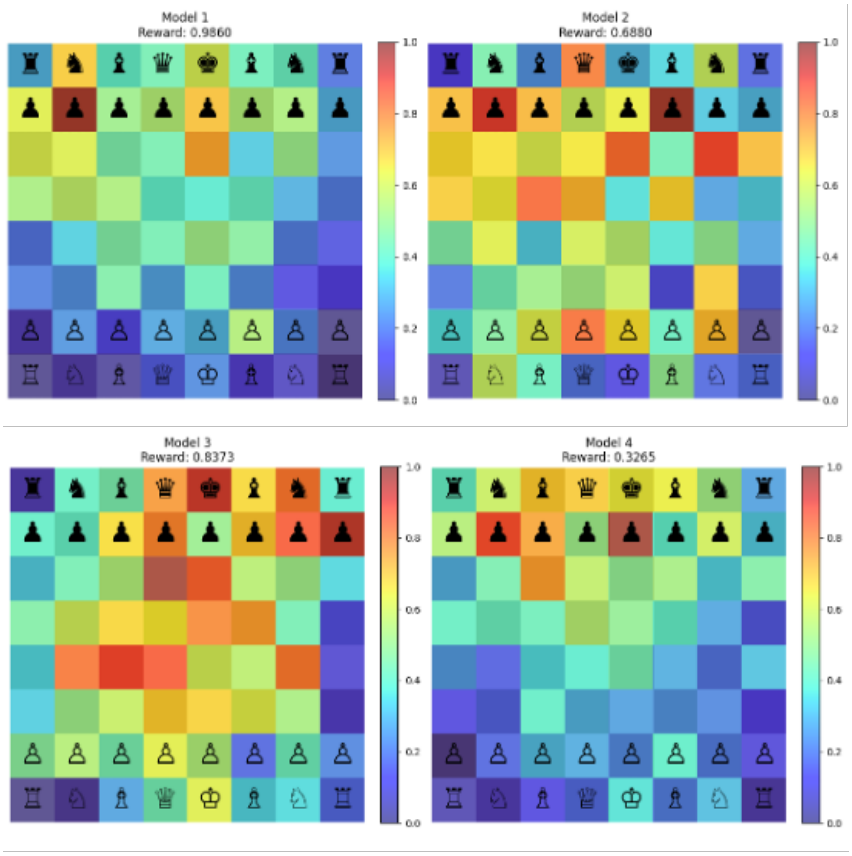

Figure 2: Reward Visualization

5 Conclusion

This project demonstrated that Maximum Likelihood Inverse Reinforcement Learning can successfully extract dense reward functions from chess demonstrations, but the quality of that reward is highly sensitive to model capacity and regularization.

The Overfitting Trap We found that "more data and bigger models" is not always better in adversarial IRL. Model 1 (2500 games, Large ResNet) collapsed into a perfect discriminator, failing to provide a shaped reward signal for learning.

Regularization is Key Model 2 trained with higher entropy and fewer blocks, learning a softer reward function with broader saliency maps, showing that constraining the discriminator forces it to learn generalizable chess concepts rather than memorizing games.

Visual vs. Calculative Style The framework was far more effective at recovering human intuition (Magnus) than engine calculation (Stockfish). This suggests that CNN-based IRL is best suited for imitating "visual" strategies rather than "computational" ones.

It would be interesting with future work to explore adding position type encoding into the reward model to track the priorities of the expert across game states.

References

[1] J. Fu, K. Luo, and S. Levine, "Learning Robust Rewards with Adversarial Inverse Reinforcement Learning," in *International Conference on Learning Representations (ICLR)*, 2018.

[2] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[3] S. Zeng, C. Li, A. Garcia, and M. Hong, "Maximum-Likelihood Inverse Reinforcement Learning with Finite-Time Guarantees," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[4] A. Y. Ng and S. Russell, "Algorithms for Inverse Reinforcement Learning," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, 2000.

A IRL Loss

Figure 3: Model 1 IRL Loss

Figure 4: Model 2 IRL Loss