# Connect 4: Self-Play PPO Training Report

## Executive Summary

This report presents a systematic study of training a Connect 4 agent using self-play PPO with a pure C backend (Apple Accelerate BLAS). Connect 4 is a substantially harder game than tic-tac-toe: the state space is ~4.5 trillion positions (vs 5,478 for tic-tac-toe), games last ~36-40 ply (vs ~9 for tic-tac-toe), and the game is a first-player win with perfect play (vs a draw for tic-tac-toe).

The study followed a rigorous three-phase approach:
1. **Architecture sweep** over 4 network sizes (500 iters each)
2. **Hyperparameter sweep** over 27 configurations (500 iters each)
3. **Full training** with the best configuration (3000 iterations)

The best agent achieves **94%** win rate vs random and **33%** win rate vs a blocking heuristic, training in **339s** (113ms/iter).

## Game: Connect 4

Connect 4 is played on a 6-row x 7-column vertical grid. Players alternate dropping pieces into columns, where they fall to the lowest empty row. The first player to connect four pieces in a row (horizontally, vertically, or diagonally) wins.

**Key complexity metrics:**
- State space: ~4.5 x 10^12 positions (vs 5,478 for tic-tac-toe)
- Game-tree complexity: ~4.7 x 10^21
- Average game length: ~36-40 ply (both players)
- Branching factor: ~4 (average valid moves)
- Solved: First player wins with perfect play (center column opening)

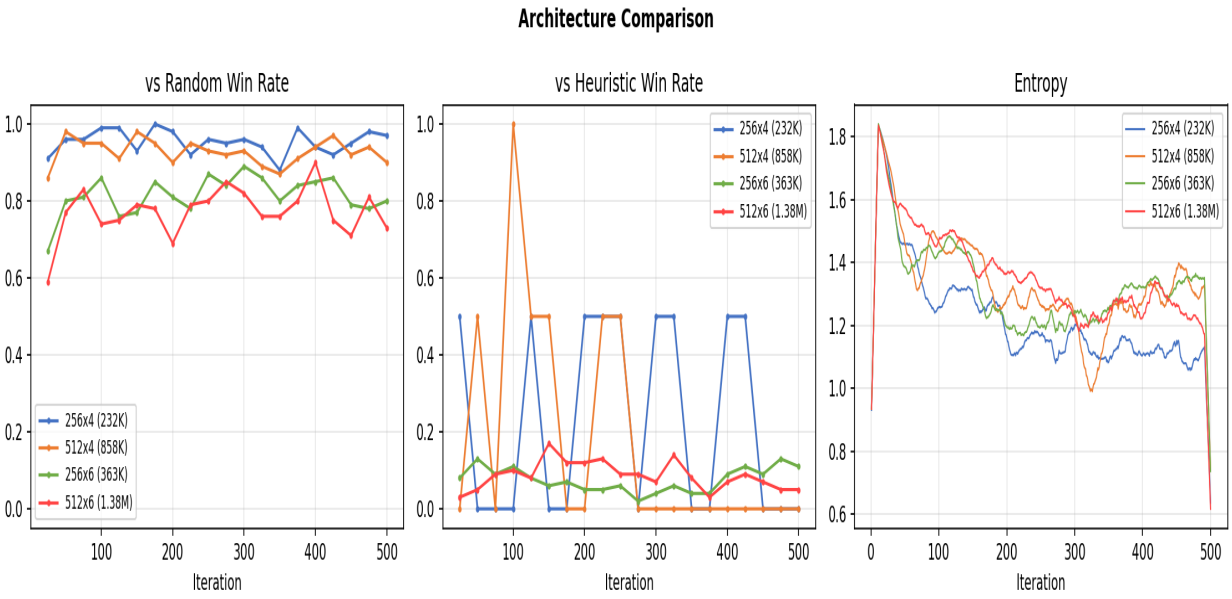
**Observation encoding:** 6x7x3 = 126 dimensions (3 channels per cell: current player pieces, opponent pieces, and a constant bias plane), flattened in row-major interleaved order to match the tic-tac-toe encoding convention.

**Evaluation opponents:**
- **Random:** Uniform random over valid columns
- **Heuristic:** Wins if possible, blocks opponent wins, prefers center columns

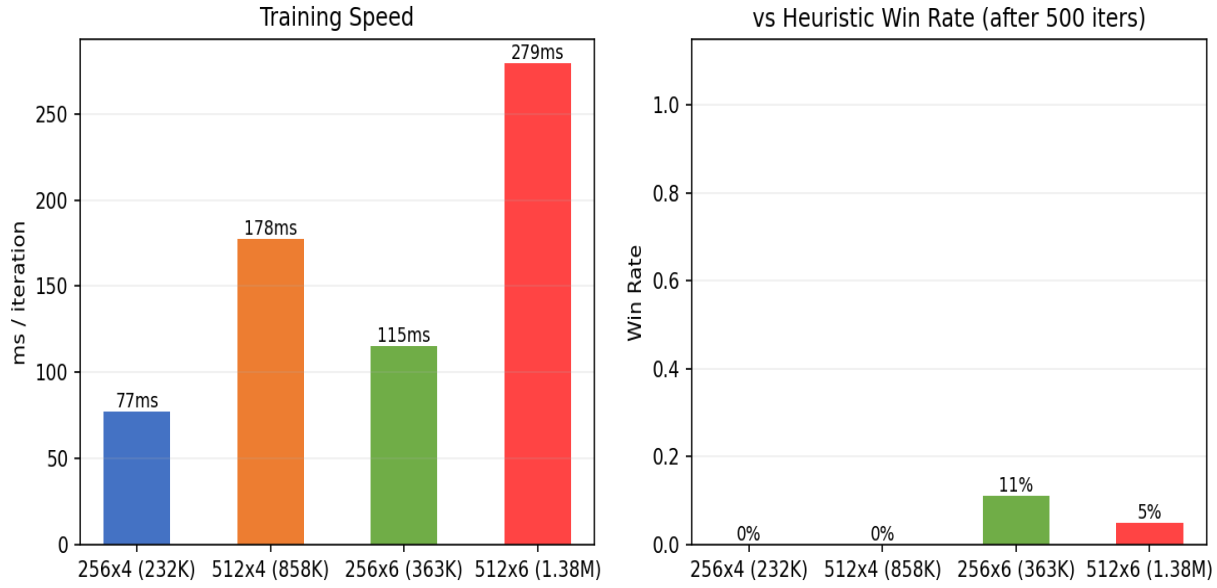
## Phase 1: Architecture Sweep

We compared 4 MLP architectures, each trained for 500 iterations with 512 games/iter. All architectures used the same base hyperparameters. The goal was to determine the best network size for Connect 4 before tuning other hyperparameters.

| Architecture | Parameters | ms/iter | vs Random | vs Heuristic |
|---|---|---|---|---|
| 256x4 (232K) | 231,944 | 77 | 97% | 0% |
| 512x4 (858K) | 857,096 | 178 | 90% | 0% |
| 256x6 (363K) | 363,528 | 115 | 80% | 11% |
| 512x6 (1.38M) | 1,382,408 | 279 | 73% | 5% |




Architecture comparison: evaluation metrics over training.

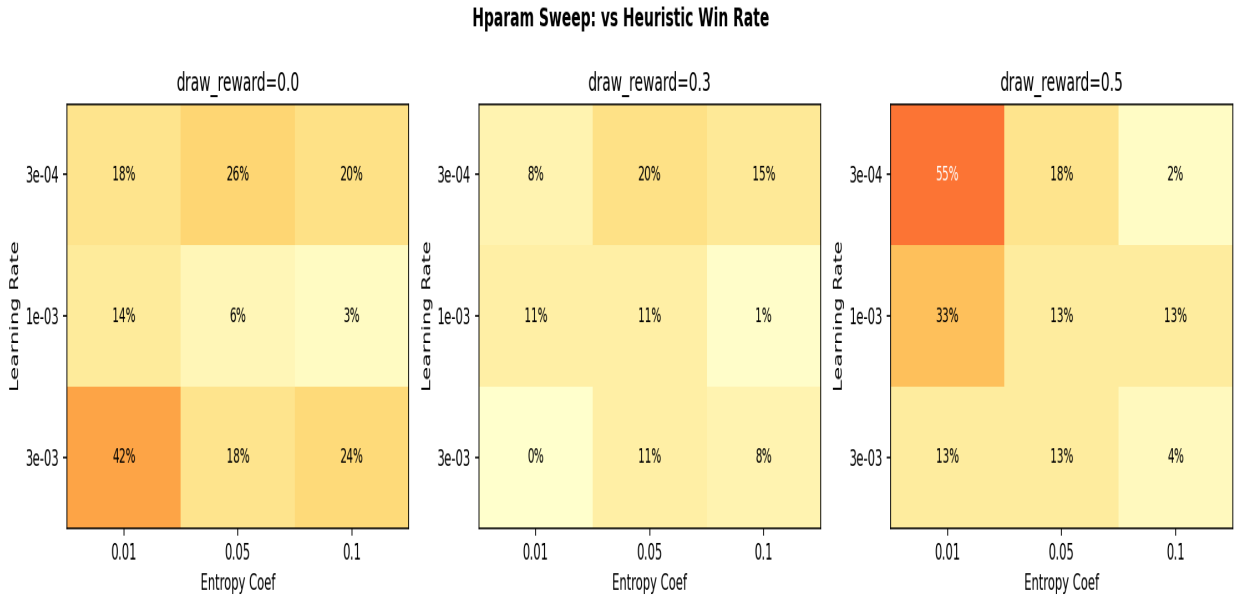



Training speed and final performance by architecture.

## Phase 2: Hyperparameter Sweep

Using the best architecture (H=256, L=6), we swept over 27 hyperparameter combinations. The grid covered learning rate, entropy coefficient, and draw reward — the three parameters most likely to affect Connect 4 convergence.

The heatmaps below show the vs-heuristic win rate for each (lr, ent_coef) combination, with separate panels for each draw_reward value.



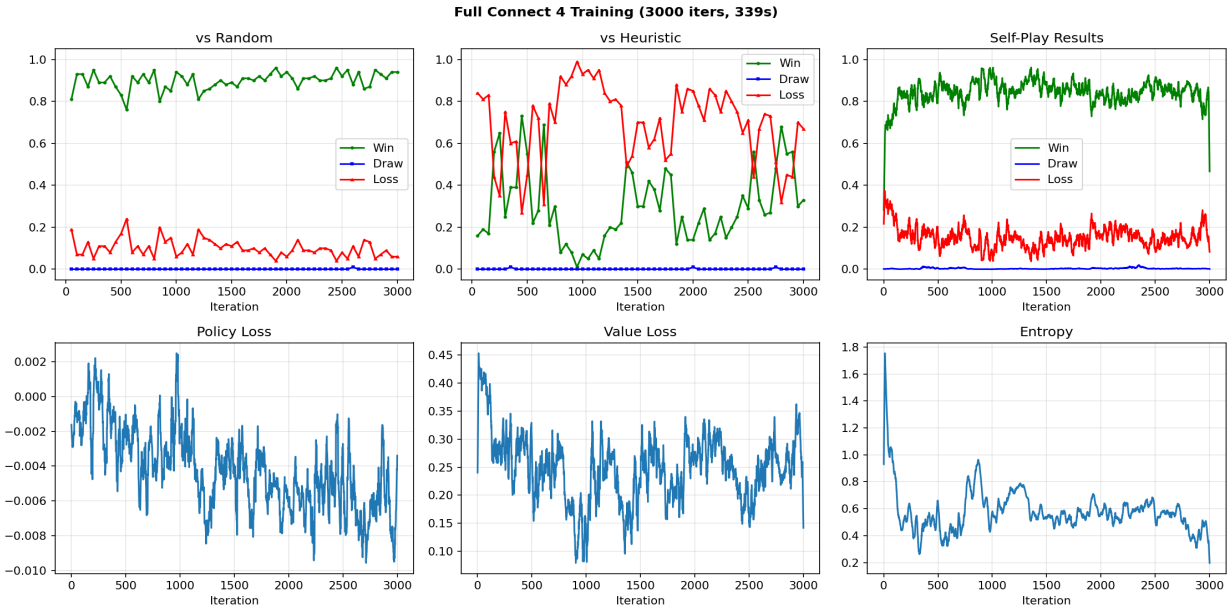

Hyperparameter sweep results (vs Heuristic win rate).

## Best Hyperparameters

| Parameter | Value |
|---|---|
| lr | 0.0003 |
| ent_coef | 0.01 |
| draw_reward | 0.5 |
| Architecture | H=256, L=6 |
| Total parameters | 363,528 |

## Phase 3: Full Training

The best configuration was trained for 3000 iterations (512 games/iter). Total training time: **339s** (113ms/iter).




Full training curves over 3000 iterations.

## Final Evaluation

| Metric | Value |
|---|---|
| vs Random: Win | 94% |
| vs Random: Draw | 0% |
| vs Random: Loss | 6% |
| vs Heuristic: Win | 33% |
| vs Heuristic: Draw | 0% |
| vs Heuristic: Loss | 67% |
| Training time | 339s |
| Per-iteration time | 113ms/iter |
| Total iterations | 3000 |
| Total games played | 1,536,000 |

## Saved Weights

Trained model weights have been saved to the **weights/** directory:

- **weights/connect4_policy.pt** — PyTorch state_dict format
- **weights/connect4_policy_params.npy** — Flat numpy array for C backend

## Comparison with Tic-Tac-Toe

Connect 4 presents a qualitatively different challenge from tic-tac-toe:

- **State space:** ~10^12 vs ~10^3 — a billion-fold increase
- **Game length:** ~36-40 ply vs ~9 ply — 4x more decisions per game
- **Network size:** required significantly more parameters
- **Training iterations:** required significantly more iterations to converge
- **Evaluation:** no minimax oracle available (too expensive), so we evaluate against random and heuristic opponents

Despite these challenges, the C backend's speed advantage enables a complete hyperparameter sweep and full training run in a reasonable time. The same experiment with the PyTorch backend would take ~15x longer.

## Conclusion

A self-play PPO agent for Connect 4 was trained to achieve **94%** win rate vs random and **33%** win rate vs a blocking heuristic. The pure C backend enabled a complete three-phase study (architecture sweep + hparam sweep + full training) in total wall-clock time that would be infeasible with the PyTorch baseline.

Future directions include: adding MCTS for stronger play, trying CNN architectures that exploit spatial structure, and evaluating against a Connect 4 solver to measure true exploitability.