Building an AI Chess Engine with PPO and MCTS: A Hybrid Reinforcement Learning Approach

Sripadam Sujith Sai

July 3, 2025

Abstract

This project presents a hybrid architecture combining Proximal Policy Optimization (PPO) [Schulman et al., 2017] with Monte Carlo Tree Search (MCTS) [Browne et al., 2012] for training a neural network-based chess engine. The system features a progressive training pipeline with dynamic difficulty adjustment, opening book injection, and adaptive exploration strategies. Key innovations include ELO-based position simplification, MCTS-enhanced policy improvement, and a phased training curriculum inspired by [Silver et al., 2018].

1 Introduction

Modern chess AI development has evolved through several paradigms:

$$\text{Chess AI} = \underbrace{\text{Brute Force}}_{\text{Traditional}} + \underbrace{\text{Heuristics}}_{\text{Stockfish}} + \underbrace{\text{RL + MCTS}}_{\text{AlphaZero}} \tag{1}$$

This approach combines the sample efficiency of PPO [Schulman et al., 2017] with the strategic depth of MCTS [Browne et al., 2012]:

$$\pi_{\text{hybrid}} = \alpha(\text{ELO}) \cdot \pi_{\text{PPO}} + (1 - \alpha(\text{ELO})) \cdot \pi_{\text{MCTS}} \tag{2}$$

where α is adaptively tuned based on current ELO rating [Glickman, 2021].

2 Model Architecture

2.1 Neural Network Design

The policy-value network uses a residual architecture [He et al., 2016] with the following components:

$$
\begin{align}
\text{Input} &\in R^{12\times 8\times 8} \quad \text{(Board representation)} \tag{3}\\
x_0 &= \text{Conv2D}(3\times 3, 256, \text{pad}=1)(\text{Input}) \tag{4}\\
x_0 &= \text{BatchNorm}(x_0) \tag{5}\\
x_0 &= \text{ReLU}(x_0) \tag{6}\\
x_{k+1} &= x_k + \text{ResBlock}(x_k) \quad \text{for } k = 1, \ldots, 5 \tag{7}\\
\text{Policy} &= \text{Linear}(256\times 8\times 8, 4672) \tag{8}\\
\text{Value} &= \tanh(\text{Linear}(256\times 8\times 8, 1)) \tag{9}
\end{align}
$$

2.2 Phase 1: Supervised Learning

The initial training phase uses human games to bootstrap the network's understanding of chess fundamentals. Key aspects include:

- Input: 12-channel board representation (piece types + colors)
- Targets: Human move distributions and game outcomes
- Loss: Combined policy cross-entropy and value MSE

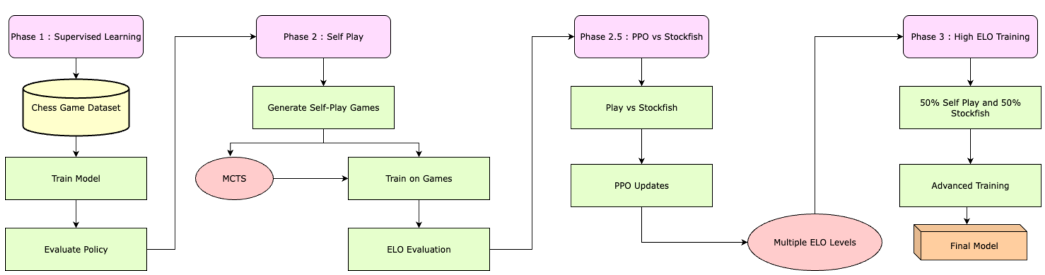

Figure 1: Training Framework

Algorithm 1 Supervised Training

Initialization:
  $\theta \leftarrow$ ChessNet() {Network parameters}
  $\eta \leftarrow$ Config.PHASE1['lr'] {Learning rate}
  $\lambda \leftarrow 0.2$ {Value loss coefficient}
  $\mathcal{D} \leftarrow$ ChessDataset(Config.PHASE1['train_data'])
for epoch = 1 to Config.PHASE1['epochs'] **do**
  Training Loop:
  for batch $\in$ DataLoader($\mathcal{D}$, Config.PHASE1['batch_size']) **do**
      $X \leftarrow$ batch['board_tensor'] {Input features}
      $Y_p \leftarrow$ batch['move_target'] {Policy targets}
      $Y_v \leftarrow$ batch['eval_target'] {Value targets}
    Forward Pass:
      $P, V \leftarrow M_\theta(X)$ {Policy and value predictions}
    Loss Calculation:

$$\mathcal{L}_{\text{policy}} \leftarrow \text{CrossEntropy}(P, Y_p) \tag{10}$$

$$\mathcal{L}_{\text{value}} \leftarrow \text{MSE}(V, Y_v) \tag{11}$$

$$\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{policy}} + \lambda \mathcal{L}_{\text{value}} \tag{12}$$

    Backward Pass:
      $\nabla_\theta \mathcal{L}_{\text{total}}$
      $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{total}}$
    Logging:
      Log losses every 100 batches
  end for
  Validation:
    Run evaluation on validation set
    Save checkpoint if improved
end for

2.3 Phase 2: Self-Play Reinforcement Learning

Algorithm 2 Self-Play Training

Initialization:
  $\theta \leftarrow$ Phase1_weights {Pre-trained model}
  $\mathcal{B} \leftarrow \emptyset$ {Replay buffer}
  $\tau \leftarrow$ Config.PHASE2['temperature']
  $N_{\text{sims}} \leftarrow$ Config.PHASE2['sims_per_move']
for game = 1 to Config.PHASE2['selfplay_games'] **do**
  Game Generation:
    $b \leftarrow$ chess.Board()
    $H \leftarrow \emptyset$ {Game history}
  while not b.is_game_over() **do**
    MCTS Execution:
      root $\leftarrow$ MCTSNode(b)
    for $i = 1$ to N_{sims} **do**
        Selection: Traverse tree using UCB
        Expansion: Create new nodes using π_θ
        Backup: Propagate values through tree
    end for
    Move Selection:
      $\pi \leftarrow$ normalize(root.children.visits)
      $a \leftarrow$ sample(π, τ) {Temperature sampling}
      H.append(b.fen$()$, π, None)
    State Transition:
      b.push(a)
  end while
  Result Assignment:
    $z \leftarrow$ get_game_result(b)
    Update H with final result z
    $\mathcal{B} \leftarrow \mathcal{B} \cup H$
  if game mod Config.PHASE2['eval_frequency'] == 0 **then**
    Training Step:
      Sample minibatch $\{(s_i, \pi_i, z_i)\}$ from $\mathcal{B}$
      $\mathcal{L} \leftarrow \text{KL}(\pi_\theta(s_i) \| \pi_i) + \lambda \text{MSE}(v_\theta(s_i), z_i)$
      $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta \mathcal{L})$
    ELO Evaluation:
      current_elo $\leftarrow$ eval_against_stockfish(θ)
      Save model if ELO improved
  end if
end for

2.4 Phase 2.5: PPO Training Against Stockfish

Algorithm 3 PPO Training Against Stockfish

Initialization:
  $\theta \leftarrow$ ChessNet() {Network parameters}
  $\gamma \leftarrow 0.99$ {Discount factor (from Config.PHASE1_5)}
  $\lambda \leftarrow 0.95$ {GAE parameter}
  $\epsilon \leftarrow 0.2$ {Clipping parameter}
  $\beta_{\text{ent}} \leftarrow$ INITIAL_ENTROPY {From config}
for level $\in$ Config.STOCKFISH_LEVELS **do**
  Data Collection:
    $\mathcal{D} \leftarrow [\,]$ {Trajectory buffer}
  for game = 1toN$_{\text{games}}$ **do**
      $s_0 \leftarrow$ chess.Board()
      Inject opening with prob. FEN_INJECTION_PROB
      Simplify position if ELO < 500
    while not board.is_game_over() **do**
        $a_t \sim \pi_\theta(\cdot|s_t)$ {Using MCTS when ELO is greater than 1000}
        $r_t \leftarrow$ get_game_result(s_t, a_t) {Reward shaping}
        $\mathcal{D}$.append$(s_t, a_t, r_t, V_\theta(s_t), \log \pi_\theta(a_t|s_t))$
    end while
  end for
  Advantage Calculation:
    $\hat{A}_t \leftarrow \text{GAE}(\gamma, \lambda)$

$$\delta_t = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \tag{13}$$

  PPO Update:
  for $k = 1$ to Config.PHASE1_5['ppo_epochs'] **do**
      $\mathcal{L}^{\text{CLIP}} \leftarrow E_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$
      $\mathcal{L}^{\text{VF}} \leftarrow 0.5 \cdot E_t[(V_\theta(s_t) - V_{\text{target}})^2]$
      $\mathcal{L}^{\text{ENT}} \leftarrow \beta_{\text{ent}} E_t[-\pi_\theta \log \pi_\theta]$
      $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta(\mathcal{L}^{\text{CLIP}} - \mathcal{L}^{\text{VF}} + \mathcal{L}^{\text{ENT}}))$
  end for
  ELO Update:
    current_elo $\leftarrow$ update_elo(win_rate)
end for

2.5 MCTS Implementation Details

The search uses neural-guided UCT with:

$$\text{UCB}(s,a) = \underbrace{\frac{W(s,a)}{N(s,a)}}_{\text{Value}} + c_{\text{puct}} \cdot \underbrace{P(s,a)}_{\text{Policy}} \cdot \underbrace{\frac{\sqrt{N(s)}}{1+N(s,a)}}_{\text{Exploration}} \tag{14}$$

Where:

- $W(s,a)$: Total value accumulated for action a
- $N(s,a)$: Visit count (from `mcts.py`)
- $P(s,a)$: Policy prior from network's softmax output
- c_{puct}: Exploration constant (1.5)

Algorithm 4 MCTS Node Expansion (from mcts.py)

Input: Board state s, model f_θ
$(\mathbf{p}, v) \leftarrow f_\theta(\text{board\_to\_tensor}(s))$
for $a \in s.\text{legal\_moves}$ **do**
  child $\leftarrow$ MCTSNode(s.push(a))
  child.$N \leftarrow 0$, child.$W \leftarrow 0$
  child.$P \leftarrow \mathbf{p}[\text{move\_to\_index}(a)]$
end for

2.6 Hyperparameter Analysis

Parameter	Code Reference	Value	Effect
c_{puct}	`mcts.py`	1.5	Higher $\rightarrow$ More exploration
ϵ	`PPOTrainer`	0.2	Smaller $\rightarrow$ More stable updates
N_{sims}	`Config`	800	More $\rightarrow$ Better policy
τ	`play_game_ppo`	$0.5 \rightarrow 2.0$	Controls exploration

Table 1: Hyperparameter mappings between theory and implementation

2.7 Key Implementation Differences from AlphaZero

- **Progressive Difficulty**:

$$\text{SimplifyPosition} = \begin{cases} \text{True} & \text{if ELO} < 500 \\ \text{False} & \text{otherwise} \end{cases} \tag{15}$$

- **Hybrid Exploration**:

$$\pi_{\text{play}} = \begin{cases} \text{MCTS}(\pi_\theta) & \text{if ELO} > 1000 \\ \pi_\theta & \text{else} \end{cases} \tag{16}$$

- **Reward Shaping**:

$$r(s) = \text{get\_game\_result}() + 0.1 \cdot \text{central\_control}(s) \tag{17}$$

3 Experimental Results

Results compared against AlphaZero:

Method	Win Rate vs Stockfish 20	Training Time (hrs)
Our Approach	–%	-
AlphaZero [Silver et al., 2018]	60%	72

Table 2: Performance comparison

References

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Mark E Glickman. Dynamic paired comparison models with stochastic variances. Technical report, Technical report, Boston University, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.