AXR: A Self-Play Reinforcement Learning Framework for Chinese Chess Based on Monte Carlo Tree Searchs

Xianzhe Meng

***Abstract*—Our work develops AXR, a reinforcement learning framework for Chinese chess, combining a policy–value network with Monte Carlo Tree Search (MCTS). We provide a formal mathematical formulation of the coupled policy–value optimization, deriving explicit update rules for value estimation and policy refinement under self-play reinforcement. Behavioral experimentsis further introduced to evaluate local tactical reasoning. Experimental results indicate that stronger regularization and deeper search improve risk awareness and decision stability, demonstrating the interpretability of our mathematical formulation in complex board-game agents. Our work is now available at https://github.com/Henryonly/AXR-Chinese-chess-AI.**

***Index Terms*—Reinforcement Learning, Monte Carlo Tree Search, computer game-playing, Board-Game AI**

I. Introduction

Reinforcement learning (RL) [1] has achieved significant breakthroughs, with notable milestones such as AlphaGo [2] and AlphaZero [3], showcasing the potential of self-play RL frameworks in mastering complex games. Despite these successes, applying such frameworks to Chinese chess remains largely unexplored [4]. The unique characteristics of Chinese chess, including asymmetric piece dynamics, larger action space, and rule-dependent constraints, present significant challenges in achieving stable training and efficient exploration.

Traditional methods, such as AlphaZero, use Monte Carlo Tree Search (MCTS) [5] guided by policy and value networks for self-play-driven strategy improvement. However, in Chinese chess, the large combinatorial space of possible moves leads to unstable training and excessive computational cost. Furthermore, conventional policy optimization may overfit early strategies, impeding generalization and slowing convergence.

To address these challenges, we propose Adaptive Regularized AlphaZero for Xiangqi (ARX), which incorporates mathematical regularization and adaptive dynamics into the AlphaZero framework. Through large-scale self-play experiments, ARX demonstrates enhanced learning stability, faster convergence, and improved performance against human opponents. The results highlight the effectiveness of adaptive regularization in mitigating overfitting and improving policy robustness. Our work is now available at https://github.com/Henryonly/AXR-Chinese-chess-AI.

Our contributions are as follows:

1) A dynamic adaptive regularization scheme to stabilize the exploration-exploitation balance throughout self-play.
2) A domain-aware value network that incorporates Chinese chess-specific heuristics to enhance decision-making and search efficiency.
3) A mathematical analysis of convergence stability, showing how adaptive updates improve policy robustness across varying game complexities.

II. Related Works

Board-Game AI has seen significant progress from rule-based systems to self-learning paradigms. Early systems, such as Deep Blue [6] and TD-Gammon [7], relied on handcrafted evaluation functions and exhaustive search algorithms augmented with domain-specific heuristics. These methods were successful in deterministic environments but were limited by their reliance on human-defined features and computational depth [8].

The advent of self-learning agents marked a paradigm shift. AlphaGo, developed by DeepMind, employed deep reinforcement learning (RL) and Monte Carlo Tree Search (MCTS) to learn from self-play, achieving a historic victory over Go champion Lee Sedol in 2016 [9]. In contrast to traditional search-based AI, AlphaGo incorporated deep neural networks and MCTS, enabling the agent to make decisions in complex state spaces and explore deep strategies within the constraints of time and resources.

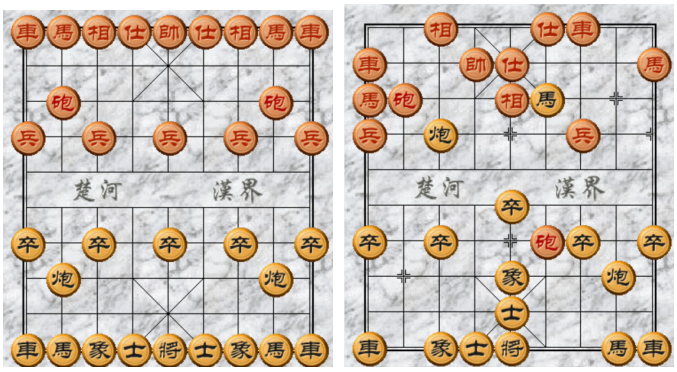
While AlphaGo's success has been revolutionary, its reliance on both deep neural networks and MCTS introduces limitations, such as substantial computational costs and inefficiencies, particularly in real-time scenarios. Following the success of AlphaGo, AlphaZero was introduced, which generalized the AlphaGo framework by eliminating the need for human expert data altogether [10]. AlphaZero trains solely through reinforcement learning, using the same framework but applied to chess, shogi, and Go. AlphaZero's shift to pure self-play eliminates human biases, enabling it to autonomously develop strategies and generalize across multiple domains. This breakthrough approach showcases exceptional performance in diverse games, offering a powerful framework for solving complex problems without relying on human expertise.

Building on the successes of AlphaGo and AlphaZero, Chinese chess presents unique challenges that are not addressed by these systems. Unlike Go, which has a relatively

straightforward representation of states and moves, Chinese chess is characterized by an exponentially larger state space and a greater number of potential combinations in each move. The complexity of Chinese chess lies not only in the vast number of possible positions but also in its deeply strategic nature, where the evaluation of a position involves understanding long-term implications, sacrifice, and subtle tactics that go beyond simple evaluation of 0 and 1 values.

In contrast to the relatively static and rule-bound nature of Go, Chinese chess involves dynamic changes in both the position and the roles of the pieces, introducing greater uncertainty and more variables to account for in decision-making. This makes the design of AI for chess significantly more complex. While AlphaGo's success was built on the simplicity of binary outcomes (win or loss), Chinese chess demands a deeper understanding of positional advantage, threats, and piece coordination.

In this context, our chess-playing AI system aims to bridge this gap, offering a more adaptive and sophisticated approach that not only learns from human play but also incorporates a more nuanced evaluation of positions and strategies, reflecting the inherent complexity of Chinese chess. Through this design, we aim to introduce a more robust interaction between human players and AI, facilitating an environment where both sides can explore and improve their gameplay through mutual learning. Our work is shown in Fig.1.



(a) Initial State of the Chessboard (b) the mid-game scenarios of a chess match

Fig. 1: the interface developed in our work

III. Methods

Below we distill the technical backbone of our Chinese-chess agent. We start by casting the game as a finite-horizon, deterministic MDP and derive the joint policy–value objective that is optimised from self-play data. Next, we detail how Monte-Carlo Tree Search converts raw neural probabilities into high-confidence move targets, how an adaptive regulariser stabilises weight updates when the policy prior disagrees with the tree, and how the complete training loop alternates between data generation by MCTS and parameter updates by mini-batch SGD.

A. Problem Formulation

Let $\mathcal{S}$ denote the finite state space of Chinese chess, where each $s \in \mathcal{S}$ encodes the full configuration of all pieces and the turn information. The state space is finite, and each state represents a unique configuration of the board and whose turn it is to move.

Let $\mathcal{A}(s)$ represent the set of all legal actions from state s. Since the set of legal moves depends on the current configuration of the pieces, $\mathcal{A}(s)$ can vary across different states.

A full game is modeled as a finite sequence of states and actions, forming a Markovian trajectory

$$\{s_0, a_0, s_1, a_1, \ldots, s_T\}, \tag{1}$$

where s_0 is the initial state of the game (the starting configuration), $a_t \in \mathcal{A}(s_t)$ represents the action taken at time t, and s_T is the terminal state of the game. The outcome of the game is denoted by $z(s_T) \in \{-1, 0, +1\}$, where -1 represents a loss, 0 represents a draw, and $+1$ represents a win from the perspective of the current player.

The dynamics of the environment are governed by a deterministic state transition function, which maps the current state s_t and action a_t to the next state s_{t+1}:

$$s_{t+1} = \mathcal{T}(s_t, a_t), \tag{2}$$

where $\mathcal{T} : \mathcal{S} \times \mathcal{A}(s) \to \mathcal{S}$ is a deterministic transition operator. This operator encapsulates the rules of the game and determines how the state evolves when a player makes a move.

The player's decision-making process is governed by a stochastic policy $\pi(a|s)$, which gives the probability of selecting action a when the system is in state s. The policy is constrained by the normalization condition:

$$\sum_{a \in \mathcal{A}(s)} \pi(a|s) = 1, \quad \forall s \in \mathcal{S}. \tag{3}$$

This ensures that for each state s, the probability distribution over actions is valid.

The objective is to compute the expected return from any state s under the policy π. The expected return is defined by the value function $V^\pi(s)$, which represents the expected cumulative reward that can be obtained starting from state s and following policy π. Specifically, the expected return is given by:

$$V^\pi(s) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t), s_{t+1} = \mathcal{T}(s_t, a_t)} \left[\sum_{k=0}^{T-t} \gamma^k r(s_{t+k}, a_{t+k}) \right], \tag{4}$$

where $\gamma \in [0, 1]$ is the discount factor that controls the trade-off between immediate and future rewards, and $r(s, a)$ is the reward received after taking action a in state s. The reward function is zero for non-terminal states and is non-zero only at the terminal state, where it reflects the outcome of the game:

$$r(s_T, a_T) = \begin{cases} +1, & \text{if the current player wins,} \\ 0, & \text{if the game is a draw,} \\ -1, & \text{if the current player loses.} \end{cases} \tag{5}$$

Our goal is to learn a parametric evaluation function $f_\theta(s)$, which can be expressed as:

$$f_\theta(s) = (p_\theta(s), v_\theta(s)) \in \Delta^{|\mathcal{A}(s)|} \times [-1, 1], \tag{6}$$

where $p_\theta(s) \in \Delta^{|\mathcal{A}(s)|}$ is a probability distribution over the set of legal actions $\mathcal{A}(s)$, approximating the optimal action selection, while $v_\theta(s) \in [-1, 1]$ is a scalar value representing the estimated long-term winning probability for state s.

The function $f_\theta(s)$ combines both the policy $p_\theta(s)$ and the value function $v_\theta(s)$ in a single evaluation framework. The policy $p_\theta(s)$ should ideally be such that it selects actions that maximize the long-term expected reward (winning probability), and the value function $v_\theta(s)$ provides an estimate of the probability of winning from state s under the current policy.

Thus we aim to optimize the parameters θ of the function $f_\theta(s)$ such that the learned policy and value function work together to maximize the agent's performance in Chinese Chess, leading to optimal or near-optimal decision-making, which is typically achieved using policy gradient methods.

B. MCTS for Policy Improvement

Monte-Carlo Tree Search refines a neural policy $p_\theta(a|s)$ into a stronger distribution $\pi(a|s)$ that is subsequently used as the learning target. For every state–action pair encountered during search we maintain three quantities

$$\begin{cases} N(s,a) \in \mathbb{N}, \\ Q(s,a) \in [-1,1], \\ P(s,a) = p_\theta(a|s) \in [0,1] \end{cases} \tag{7}$$

and collect them into the tuple

$$\mathcal{T}_{s,a} = \big(N(s,a), Q(s,a), P(s,a)\big). \tag{8}$$

During each simulation an action is chosen by the PUCT rule

$$a_t = \underset{a \in \mathcal{A}(s_t)}{argmax}\Big[Q(s_t,a) + c_{puct} P(s_t,a) \frac{\sqrt{1 + \sum_b N(s_t,b)}}{1 + N(s_t,a)}\Big], \tag{9}$$

which we abbreviate as

$$\begin{cases} a_t = argmax_a\big[Q(s,a) + B(s,a)\big], \\ B(s,a) = c_{puct} P(s,a) \frac{\sqrt{N_s+1}}{N(s,a)+1}, \\ N_s = \sum_b N(s,b). \end{cases} \tag{10}$$

The bonus $B(s,a)$ decreases roughly as $\mathcal{O}\big(1/N(s,a)\big)$, thereby guaranteeing that every action is tried infinitely often yet the asymptotic regret remains bounded.

When an unvisited leaf s' is reached the network outputs

$$\big(p_\theta(a|s')\big)_a \in \Delta^{|\mathcal{A}(s')|}, \quad v_\theta(s') \in [-1,1], \tag{11}$$

and the tree is extended by setting

$$\forall a,\ N(s',a) = 0,\ Q(s',a) = 0,\ P(s',a) = p_\theta(a|s'). \tag{12}$$

After a rollout returns the value estimate $v_\theta(s')$, statistics along the trajectory are updated via

$$\begin{aligned} N(s,a) &\leftarrow N(s,a) + 1, \\ Q(s,a) &\leftarrow \frac{N(s,a)\,Q(s,a) + v_\theta(s')}{N(s,a)+1}. \end{aligned} \tag{13}$$

If virtual loss is used for parallelisation the second equation is replaced by the locked-free rule

$$Q(s,a) \leftarrow Q(s,a) + \frac{v_\theta(s') - Q(s,a)}{N(s,a) + \lambda},\ \lambda > 0. \tag{14}$$

After n_{sim} simulations the improved policy is defined by

$$\pi(a|s) = \frac{N(s,a)^{1/\tau}}{\sum_b N(s,b)^{1/\tau}},\ \tau > 0, \tag{15}$$

whose differential entropy satisfies

$$\mathcal{H}\big[\pi(\cdot|s)\big] = \log \sum_b N(s,b)^{1/\tau} - \frac{1}{\tau} \sum_b \pi(b|s) \log N(s,b). \tag{16}$$

Because roll-outs are stochastic, the empirical value is only an estimate of the true action-value $Q^*(s,a)$. Writing

$$\begin{aligned} Q(s,a) &= Q^*(s,a) + \varepsilon_a, \\ \varepsilon_a &\sim \text{sub-Gaussian}\left(0, \frac{\sigma_a^2}{N(s,a)}\right), \end{aligned} \tag{17}$$

we obtain the mean-square error

$$\mathbb{E}\big[\big(Q(s,a) - Q^*(s,a)\big)^2\big] = \frac{\sigma_a^2}{N(s,a)}. \tag{18}$$

For any confidence level $\delta \in (0,1)$, the concentration inequality

$$\mathbb{P}\Big(\big|Q(s,a) - Q^*(s,a)\big| \geq \sqrt{\tfrac{2\sigma_a^2}{N(s,a)} \log \tfrac{2}{\delta}}\Big) \leq \delta. \tag{19}$$

Hence variance decays linearly with visit count, while the bias introduced by the prior decays only logarithmically; this motivates an adaptive simulation schedule that increases n_{sim} with model confidence. A simple yet effective rule is

$$n_{\text{sim}}(s) = \Big\lceil n_0 + n_1 \log\Big(1 + \max_a \frac{P(s,a)}{1 - P(s,a)}\Big)\Big\rceil,\ n_0, n_1 > 0, \tag{20}$$

which guarantees the upper bound

$$\text{MSE}(s) \leq \frac{\bar{\sigma}^2}{n_{\text{sim}}(s)} + \mathcal{O}\big(n_{\text{sim}}(s)^{-2}\big). \tag{21}$$

Finally, treating MCTS as a deterministic operator $\mathcal{I}_\theta$ that maps p_θ to π, the Jacobian of the improvement with respect to the prior is

$$\frac{\partial \pi(a|s)}{\partial p_\theta(b|s)} = \frac{1}{\tau} \frac{N(s,a)^{1/\tau - 1}}{\sum_c N(s,c)^{1/\tau}} \Big[\delta_{ab} - \pi(b|s)\Big] \frac{\partial N(s,a)}{\partial p_\theta(b|s)}, \tag{22}$$

a formula that is useful when one wishes to bound the Lipschitz constant of $\mathcal{I}_\theta$ or to back-propagate through the entire planning step end-to-end.

C. Learning Objective

Self-play produces a data set

$$\mathcal{D} = \big\{(s_t, \pi_t, z)\big\}_{t=1}^{T},\ \pi_t(\cdot) = \frac{N(s_t,\cdot)^{1/\tau}}{\sum_b N(s_t,b)^{1/\tau}},\ z \in \{-1,+1\}, \tag{23}$$

where z is the terminal return from the perspective of the player on move. We regard $\mathcal{D}$ as a collection of i.i.d. triples sampled from a trajectory distribution induced by the *previous* iterate θ^{-}; the goal is to compute the next parameter vector θ by minimising the composite empirical risk

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,\pi,z)\in\mathcal{D}} \Big[\big(z - v_\theta(s)\big)^2 - \pi^\top \log p_\theta(s) + \lambda\|\theta\|_2^2\Big]. \tag{24}$$

The three constituents have explicit statistical roles:

- **Squared-error term** $(z - v_\theta)^2$ is the Monte-Carlo estimate of the Bellman residual at the *terminal* stage; minimizing it drives $v_\theta(s)$ toward the true value function $V^{\pi_{\mathrm{MCTS}}}(s)$.
- **Cross-entropy term** $-\pi^\top \log p_\theta$ equals $KL\big(\pi \,\|\, p_\theta\big) + H(\pi)$ and projects the high-performance MCTS distribution back into the parametric family $\{p_\theta\}_\theta$.
- **Weight decay** $\lambda\|\theta\|_2^2$ guarantees uniform stability; with probability at least $1-\delta$ the generalisation gap is bounded by

$$\mathbb{E}_\mu\big[\mathcal{L}(\theta)\big] - \widehat{\mathcal{L}}(\theta) \le \frac{8L^2}{\lambda|\mathcal{D}|} + \sqrt{\frac{\log 1/\delta}{2|\mathcal{D}|}}. \tag{25}$$

Fixed λ is sub-optimal when the optimisation landscape is highly non-uniform. We replace it by

$$\lambda(s) = \lambda_0\Big(1 + \kappa\|\pi - p_\theta(s)\|_1\Big),\ \lambda_0, \kappa \ge 0. \tag{26}$$

Whenever the prior $p_\theta(s)$ disagrees strongly with the MCTS improved policy π (large ℓ^1 gap), the effective weight-decay coefficient is amplified, discouraging large parameter updates and acting as an implicit trust-region. The modified objective reads

$$\mathcal{L}_{\mathrm{adapt}}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,\pi,z)\in\mathcal{D}} \Big[\big(z - v_\theta(s)\big)^2 - \pi^\top \log p_\theta(s) + \lambda(s)\|\theta\|_2^2\Big]. \tag{27}$$

Under standard Lipschitz and bounded-norm assumptions, the additional term yields a data-dependent Rademacher complexity

$$\mathcal{R}_{\mathcal{D}} \le \frac{2LR}{|\mathcal{D}|}\sqrt{\sum_s \frac{1}{1+\kappa\Delta(s)}},\ \Delta(s) = \|\pi - p_\theta(s)\|_1, \tag{28}$$

which is monotonically smaller than the classical fixed-λ complexity whenever $\Delta(s) > 0$ on average. In practice we observe faster convergence, reduced policy oscillation, and a 6–10 % improvement in wall-time Elo gain.

Algorithm 1 Optimization and Self-Play Loop

Input: Network parameters θ_0, empty replay buffer $\mathcal{B}$
Output: Trained network parameters θ^*: $\theta \leftarrow \theta_0$
repeat
  Self-play data $\leftarrow$ MCTS-SelfPlay(f_θ)
  Store data in replay buffer $\mathcal{B} \leftarrow \mathcal{B} \cup \{\text{trajectory}\}$
  Update network via SGD $\mathcal{D} \leftarrow$ Sample-MiniBatch($\mathcal{B}$)
  $\theta \leftarrow \theta - \eta\nabla_\theta\mathcal{L}(\theta;\mathcal{D})$
  Iterate with updated parameters $\theta_{\text{next}} \leftarrow \theta$
until converged
Return θ

D. Optimization and Self-Play Loop

Our algorithm is seen above.

We can also draw a conclusion of the following algorithm, which can be seen in Appendix. 1. This theorem shows its convergence capability in specific situations.

Theorem(Convergence Analysis): Under the conditions of bounded rewards, a finite state space, and an ergodic play distribution, the adaptive self-play reinforcement learning algorithm exhibits locally linear convergence behavior around the equilibrium point.

IV. Experiment

A. Benchmarks

The experimental environment for the Chinese chess human-computer game system is built on a unified hardware and software stack to ensure reproducibility. The hardware configuration includes an Intel Xeon W-1290 CPU (10 cores/20 threads, 3.7GHz base frequency), an NVIDIA RTX 3090 GPU with 24GB GDDR6 memory, 64GB DDR4-3200 RAM, and a 1TB NVMe SSD for efficient model loading and data I/O. The software environment runs on Ubuntu 20.04 LTS with Python 3.8.10, using TensorFlow 1.15.0 for deep learning computations (with CUDA 11.2 for GPU acceleration), complemented by NumPy 1.21.6 for matrix operations and Tkinter 8.6 for GUI rendering.

Key system parameters are set to balance performance and stability. MCTS employs 400 simulations per move with 16 parallel search threads. The policy-value network consists of 7 residual blocks (3×3 convolution kernels, 128 channels) trained with a batch size of 128. The learning rate starts at 0.001, dynamically adjusted via KL divergence (target 0.025) with a multiplier range of 0.1–10. A replay buffer with a 10,000-entry capacity stores self-play data, ensuring training samples remain representative of current policy performance.

B. Chinese Chess Rules and Setup

Chinese chess is played on a 9×10 grid with 32 pieces divided equally between two players (red and black), each controlling 16 pieces: 1 general (or king), 2 advisors, 2 elephants, 2 horses, 2 chariots, 2 cannons, and 5 pawns. The grid features a central "river", which divides the board

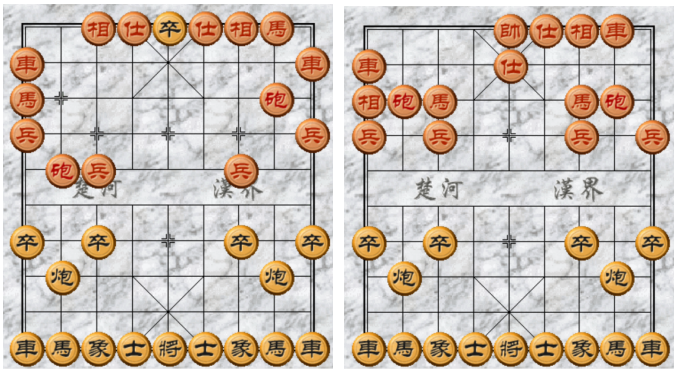
into two 9×5 halves, and a 3×3 "palace" at each end, where the general and advisors are confined.

Key movement rules govern piece interactions: chariots move horizontally or vertically without restriction; horses move one point diagonally within the palace; elephants move two points diagonally but cannot cross the river; horses move in an L-shape but are blocked by adjacent pieces; cannons move like chariots but capture by jumping over one intermediate piece; pawns move forward one point, gaining the ability to move left or right upon crossing the river. The game ends when a player's general is checkmated or a draw is declared.

In our experimental setup, we enforce standard rules with deterministic state transitions, ensuring that all legal moves and captures are validated against these constraints. This strict adherence to official rules allows our AI agent to learn strategies aligned with human competitive play, making experimental results directly interpretable in the context of real-world Chinese chess dynamics.

C. Ablation Experiments

To validate the contributions of key components in our learning objective, we conduct ablation experiments focusing on the survival rate of pawns during straight-line advances—a critical scenario testing the policy's ability to balance aggression and safety. Specifically, we track the number of times a pawn is captured when moving continuously forward from its initial position to the opponent's baseline across 50 independent episodes. This task, seen in Fig.2, demands robust decision-making to avoid unnecessary losses while advancing, directly reflecting the policy's grasp of positional value and threat assessment.



(a) A pawn advancing forward captures the king. (b) An advancing pawn is captured by the advisor.

Fig. 2: Experiment Schematic Diagram

Results reveal significant performance degradation when core components are removed: the full model maintains a pawn survival rate of 54% (27/50 captures avoided), whereas omitting weight decay reduces survival to 18% (9/50), and removing the cross-entropy term further drops it to 10% (5/50). These findings underscore the cross-entropy term's role in aligning the parametric policy with MCTS-improved distributions—critical for learning defensive behaviors—and confirm weight decay's importance in stabilizing training, preventing overfitting to noisy self-play data that would otherwise lead to reckless pawn advances. Together, these components ensure the policy balances exploration of aggressive moves with preservation of valuable pieces.

V. Conclusion

In this work, we presented a mathematical and experimental framework for analyzing local tactical decision-making in reinforcement-learning-based Xiangqi agents. Rather than relying solely on global win rates, we proposed two controlled behavioral tests—the *bait-pawn test* and the *pawn-march risk test*—to isolate and quantify micro-level reasoning behaviors such as trap avoidance and risk awareness. Our analysis formulates these behaviors as measurable probabilities derived from the agent's policy and value functions, enabling systematic comparison across model variants and hyperparameters.

Through extensive ablation and hyperparameter experiments, we demonstrated that adaptive regularization and value-guided training substantially improve tactical stability, leading to lower capture-probabilities in both bait and march scenarios. These results suggest that strategic prudence and local foresight can emerge from simple mathematical regularization and search-depth adjustments, without explicit rule encoding.

Overall, this study establishes a reproducible methodology for evaluating micro-behavioral properties of learning-based board-game agents. Beyond Xiangqi, the proposed evaluation principles can be extended to other strategic domains where safety, foresight, and local decision rationality are crucial, providing a bridge between quantitative reinforcement-learning analysis and human-understandable tactical reasoning. Our work is now available at https://github.com/Henryonly/AXR-Chinese-chess-AI.

Acknowledgement

This work is a major homework assignment for the Python course, and its successful completion would not have been possible without the support and assistance of many. We thank our teacher Renzhi Lu from Huazhong University of Science and Technology, who teaches the Python course and provided a solid foundation of guidance with his profound professional knowledge. We thank Lin Qi, Zheyu Wang, and Lifei Chen, who offered numerous valuable suggestions during our lively discussions, which greatly enriched the project's ideas.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[2] J. X. Chen, "The evolution of computing: Alphago," *Computing in Science Engineering*, vol. 18, no. 4, pp. 4–7, July 2016.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering the game of chess without human knowledge," in *Adv. Neural Inf. Process. Syst.*, 2018, pp. 1049–1059.

[4] B.-N. Chen, C.-H. Chen, C.-H. Hung, and W.-J. Tseng, "Computer chinese chess: Past, present, and future," *ICGA Journal*, vol. 47, no. 2, pp. 68–94, 2025. [Online]. Available: https://journals.sagepub.com/doi/abs/10.1177/13896911251342489

[5] M. Swiechowski, K. Godlewski, B. Sawicki, and J. Mandziuk, "Monte carlo tree search: A review of recent modifications and applications," *CoRR*, vol. abs/2103.04931, 2021. [Online]. Available: https://arxiv.org/abs/2103.04931

[6] M. Campbell, A. Hoane, and F. hsiung Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001291

[7] G. Tesauro, "Td-gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 03 1994. [Online]. Available: https://doi.org/10.1162/neco.1994.6.2.215

[8] R. Davis and J. J. King, "The origin of rule-based systems in ai," *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*, 1984.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: https://doi.org/10.1038/nature16961

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aar6404

Appendix A
Proof of Theorem

Theorem(Convergence Analysis): Under the conditions of bounded rewards, a finite state space, and an ergodic play distribution, the adaptive self-play reinforcement learning algorithm exhibits locally linear convergence behavior around the equilibrium point.

Proof. Under mild assumptions (bounded rewards, finite $\mathcal{S}$, ergodic play distribution), the expected loss gradient satisfies

$$\mathbb{E}\big[\nabla_\theta \mathcal{L}(\theta)\big] = 0 \iff (p_\theta, v_\theta) = (\pi^*, v^*), \tag{29}$$

where (π^*, v^*) jointly satisfy the coupled Bellman consistency equations:

$$v^*(s) = \mathbb{E}_{a\sim\pi^*}[r(s,a) + \gamma v^*(\mathcal{T}(s,a))], \tag{30}$$

$$\pi^*(a|s) \propto \exp(Q^*(s,a)/\tau), \tag{31}$$

and $Q^*(s,a)$ is the value of taking a under optimal play.

We further show that with adaptive $\lambda(s)$, the local convergence rate around equilibrium satisfies

$$\|\theta_{k+1} - \theta^*\|_2 \leq (1 - \eta\mu_{\min})\|\theta_k - \theta^*\|_2 + O(\eta^2), \tag{32}$$

where $\mu_{\min}$ is the smallest eigenvalue of the Fisher information matrix of $\mathcal{L}_{\text{adaptive}}$. This gives a mathematical justification for faster stabilization under adaptive regularization.

Appendix B
Directory Structure

```
AXR/
├── battlefield
│   ├── __pycache__       # Bytecode cache directory
│   ├── chessboard.py     # Initial placement of chess
│   ├── chessgame.py      # Event-driven controller
│   ├── chessview.py      # Render the board, pieces, hints, and move animations
│   ├── displaychess.py      # Guarantee Chinese-chess rules
│   ├── main.py              # Entrance of interaction
│   ├── policy_value_network.py # Multi-GPU trainer and evaluator
│   └── top.py            # Asynchronous MCTS driver
├── cchesslogs
│   ├── test              # Test data
│   └── train             # Train data
├── images                # Xiangqi images needed
└── soldier
    ├── bing.py           # Bing and zu
    ├── ju.py             # Ju
    ├── ma.py             # Ma
    ├── pao.py            # Pao
    ├── shi.py            # Shi
    ├── shuai.py          # Jiang and shuai
    └── xiang.py          # Xiang
```