Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Jiu fusion artificial intelligence (JFA): a two-stage reinforcement learning model with hierarchical neural networks and human knowledge for Tibetan Jiu chess*#

Xiali LI†‡[1,2], Xiaoyu FAN[1,2], Junzhi YU†‡[3,4], Zhicheng DONG[5], Xianmu CAIRANG[5], Ping LAN[5]

[1]*Key Laboratory of Ethnic Language Intelligent Analysis and Security Governance of MOE, Minzu University of China, Beijing 100081, China*

[2]*School of Information Engineering, Minzu University of China, Beijing 100081, China*

[3]*State Key Laboratory for Turbulence and Complex Systems, Peking University, Beijing 100871, China*

[4]*State Key Laboratory of General Artificial Intelligence, Peking University, Beijing 100871, China*

[5]*School of Information Science and Technology, Xizang University, Lhasa 850000, China*

†E-mail: lixiali@muc.edu.cn; junzhi.yu@ia.ac.cn

Received May 2, 2025; Revision accepted Sept. 15, 2025; Crosschecked Oct. 13, 2025

Abstract: Tibetan Jiu chess, recognized as a national intangible cultural heritage, is a complex game comprising two distinct phases: the layout phase and the battle phase. Improving the performance of deep reinforcement learning (DRL) models for Tibetan Jiu chess is challenging, especially given the constraints of hardware resources. To address this, we propose a two-stage model called JFA, which incorporates hierarchical neural networks and knowledge-guided techniques. The model includes sub-models: strategic layout model (SLM) for the layout phase and hierarchical battle model (HBM) for the battle phase. Both sub-models use similar network structures and employ parallel Monte Carlo tree search (MCTS) methods for independent self-play training. HBM is structured as a hierarchical neural network, with the upper network selecting movement and jump capturing actions and the lower network handling square capturing actions. Human knowledge-based auxiliary agents are introduced to assist SLM and HBM, simulating the entire game and providing reward signals based on square capturing or victory outcomes. Additionally, within the HBM, we propose two human knowledge-based pruning methods that prune parallel MCTS and capture actions in the lower network. In the experiments against a layout model using the AlphaZero method, SLM achieves a 74% win rate, with the decision-making time being reduced to approximately 1/147 of the time required by the AlphaZero model. SLM also won the first place at the 2024 China National Computer Game Tournament. HBM achieves a 70% win rate when playing against other Tibetan Jiu chess models. When used together, SLM and HBM in JFA achieve an 81% win rate, comparable to the level of a human amateur 4-dan player. These results demonstrate that JFA effectively enhances artificial intelligence (AI) performance in Tibetan Jiu chess.

Key words: Games; Reinforcement learning; Tibetan Jiu chess; Separate two-stage model; Self-play; Hierarchical neural network; Parallel Monte Carlo tree search

https://doi.org/10.1631/FITEE.2500287 **CLC number:** TP18

1 Introduction

Tibetan Jiu chess is a complete information game, encompassing both layout and battle phases (see Section 1 in the supplementary materials). It is recognized as a national intangible cultural heritage

‡ Corresponding authors

* Project supported by the National Natural Science Foundation of China (Nos. 62276285 and 62236011)

Electronic supplementary materials: The online version of this article (https://doi.org/10.1631/FITEE.2500287) contains supplementary materials, which are available to authorized users

ORCID: Xiali LI, https://orcid.org/0000-0001-7950-6204; Junzhi YU, https://orcid.org/0000-0002-6347-572X

and an exemplary traditional cultural symbol of the Chinese nation. The research on its game algorithms not only provides an ideal experimental platform for advancing artificial intelligence (AI) algorithms, but also fosters innovative approaches for the preservation and transmission of traditional culture.

Deep reinforcement learning (DRL) algorithms have been widely applied to board games, with the most achieving performance that surpasses top human players, such as AlphaGo (Silver et al., 2016), AlphaZero (Silver et al., 2018), Libratus (Brown and Sandholm, 2018), and Suphx (Li JJ et al., 2020). The outstanding performance of these agents relies on substantial hardware resources for model training. DRL algorithms for Tibetan Jiu chess include temporal-difference reinforcement learning algorithms, hybrid DRL models, and phased game strategies. However, the aforementioned algorithms or models have not achieved breakthrough progress in enhancing the playing strength of the agents, and the research on Tibetan Jiu chess game strategies faces the following three challenges:

1. For most complete information games, DRL algorithms combined with Monte Carlo tree search (MCTS) are commonly employed, where the evaluation results of the game tree guide the training of the neural network. In Tibetan Jiu chess, the game consists of layout and battle phases, with simulations extending from leaf nodes to the end of the game. This leads to excessively long search paths in the game tree. Furthermore, using a single model throughout the entire gameplay process makes it challenging for the agent to learn accurately, thereby hindering improvements in playing strength.

2. Tibetan Jiu chess lacks large-scale, high-quality training datasets. Employing self-play methods to generate training data is time-consuming under the limited hardware resources typically available in laboratory environments, making it difficult to rapidly enhance the agent's performance.

3. Currently proposed phased models adopt an improved upper confidence bounds applied to trees (UCT) algorithm for the layout phase and DRL algorithms for the battle phase. However, the Monte Carlo simulations in the layout phase extend only to the end of the layout phase rather than the actual game outcome. As a result, rewards are determined solely by evaluating the board state at the end of the layout phase, leading to inaccuracies in reward design.

To address the aforementioned three challenges, this paper presents a novel and efficient Tibetan Jiu chess model, Jiu fusion artificial intelligence (JFA), which is composed of two independently trained neural network models. One model is specifically designed for the layout phase, and is responsible for generating layout strategies, referred to as strategic layout model (SLM). The other model is designed for the battle phase, and is responsible for generating battle strategies, referred to as the hierarchical battle model (HBM). SLM and HBM share similar network structures and both employ parallel MCTS; however, their reward designs are independent, and the training of each model is also performed independently. We introduce a human knowledge-based agent, called the auxiliary agent, to support the independent training of SLM and HBM, as well as to provide rewards for SLM.

To reduce the action space in the battle phase and improve decision-making capability, HBM is designed as a hierarchical neural network, with the upper network responsible for move and jump capturing and the lower network responsible for square capturing. Both the upper and lower networks are trained independently. Furthermore, two human knowledge-based pruning methods are proposed: one for pruning the nodes of the parallel MCTS in HBM and the other for pruning the candidate capturing actions in the lower network.

Although SLM and HBM are trained independently, the training of each model is divided into two stages. At the first stage, the models are trained using data from 1700 human games to train the upper networks of both SLM and HBM. At the second stage, self-play-generated data are used for training the models, with the auxiliary agent supporting self-play training. For SLM, the auxiliary agent not only simulates decisions for the battle phase, but also provides rewards during the self-play process. For HBM, the auxiliary agent assists in the layout decision-making process and helps train both the upper and lower networks independently. During self-play for the upper network in HBM, the auxiliary agent is responsible for the "square capturing" operation, while it assists with move operations in the lower network.

Against other high-level Tibetan Jiu chess models in the actual games, both SLM and HBM outperform other models in terms of decision-making level

and decision speed. After assembling the trained SLM and HBM into JFA, its performance is even more outstanding.

2 Background

2.1 Introduction to Tibetan Jiu chess

In Tibetan Jiu chess competitions, a 14×14 chessboard is used. Players alternate placing and moving pieces, and the game consists of two main phases:

1. Layout phase. Both sides take turns placing one piece per move until the board is full.

2. Battle phase. All diagonal-line pieces are removed, and players alternate moving or capturing with their remaining pieces.

Placement rules, movement and capturing mechanics (single-step movement, jump capturing, and square capturing), classic formations (Dalian), and victory conditions are detailed in Section 2 in the supplementary materials.

2.2 Related works on DRL in games

DRL has been widely applied in games. Deep Q network (DQN) (Mnih et al., 2013) and soft actor–critic (SAC) (Haarnoja et al., 2018) have demonstrated outstanding performance in Atari games. NDP_DQN and NDP51_DQN (Zhang Z et al., 2023) have significantly improved the performance of Atari games in high-dimensional state spaces. SampleViz (Liang et al., 2024) optimizes the reward function and strategy in Atari games. Adaptive prediction sample network (APSN) (Chu, 2024) addresses the low sample efficiency in deep Q-learning by enabling efficient sample selection. Mask-Attention A3C (Itaya et al., 2024) combines the mask attention mechanism with the actor–critic method, enhancing the interpretability of agent decisions in Atari games. The Rainbow (Hessel et al., 2018) algorithm integrates DQN and its variants to resolve the inefficiencies in the classic DQN algorithm.

AlphaStar (Vinyals et al., 2019), using multi-agent reinforcement learning and self-play, outperforms human professional players in StarCraft II. OpenAI Five (OpenAI, 2019) defeats the champion team of Dota 2 through self-play reinforcement learning (RL). Libratus (Brown and Sandholm, 2018) and Pluribus (Brown and Sandholm, 2017) defeat top human players in heads-up and multiplayer poker, respectively. In the domain of Mahjong, Suphx (Li JJ et al., 2020), developed by Microsoft Research Asia, demonstrates a performance comparable to or surpassing that of top human players. AlphaGo (Silver et al., 2016) and AlphaGo Zero (Silver et al., 2017) defeat top human Go players, while AlphaZero (Silver et al., 2018) achieves great success in Go, chess, and shogi. Extensive, lightweight, and flexible (ELF) Go (Tian et al., 2017) achieves a high level of performance with lower computational requirements compared to AlphaZero.

2.3 Related works on Tibetan Jiu chess

The game algorithms for Tibetan Jiu chess encompass knowledge-based methods (Li XL et al., 2018; Zhang XC et al., 2021), DRL models (Li XL et al., 2020, 2024; Wang YJ et al., 2022; Wang SY, 2023), and phased computer game algorithms (Li XL et al., 2023).

Knowledge-based methods (Li XL et al., 2018; Zhang XC et al., 2021) typically extract chess formations from game records of human experts, and design corresponding offensive and defensive strategies for the layout and battle phases. However, these methods are limited by their reliance on human knowledge, resulting in insufficient flexibility in gameplay and restricting decision-making to predefined chess formations. Nevertheless, the perception-feedback mechanism relies heavily on human knowledge, adversely affecting the adaptability of the algorithms.

DRL models (Li XL et al., 2020, 2024; Wang YJ et al., 2022; Wang SY, 2023) generally employ self-play DRL algorithms. Li XL et al. (2020) and Wang SY (2023) optimized models through self-play training from scratch; however, training efficiency is low due to hardware resource limitations, resulting in slow progress during the self-play process. Li XL et al. (2024) introduced a lightweight U-Net network model (originally designed for Tibetan Go). According to the authors' subsequent validation, when this architecture is adapted to the 8×8 Tibetan Jiu chess board, it effectively improves the training and self-play efficiency. The cited literature employs an AlphaZero style from scratch self-play strategy, whereas the differences introduced by JFA are detailed in Section 3.6. Furthermore, the lightweight U-Net experiments are confined to an 8×8 simplified board,

and its performance on a 14×14 board has not yet been thoroughly validated. Finally, although the U-Net model achieves faster self-play, its ultimate win rate and decision-making quality remain limited and do not match the overall performance of JFA. Wang YJ et al. (2022) optimized the traditional UCT algorithm through parallel multiprocessing and combined it with a neural network model to improve the search efficiency of the game tree.

Phased computer game algorithms (Li XL et al., 2023) are designed to address the distinct phases of Tibetan Jiu chess. Li XL et al. (2023) applied a Gaussian distribution combined with a fast online estimation method during the layout phase to provide layout rewards based on the chess formation scores and used a neural network model to guide MCTS during the battle phase, demonstrating the method feasibility. Nevertheless, the reward evaluation in the layout phase remains insufficiently accurate, and the self-play speed is relatively low.

Therefore, under limited hardware resources, enhancing the self-play efficiency of DRL models, designing accurate reward mechanisms, and optimizing the extensive action space are critical challenges in improving the playing strength of Tibetan Jiu chess agents.

2.4 Related works on parallel MCTS

MCTS is a simulation-based tree search algorithm (Perez et al., 2014). Since AlphaGo integrated deep learning with MCTS to achieve superhuman performance in Go, MCTS has been increasingly applied in the field of machine game playing, prompting more in-depth research into parallel MCTS.

Chaslot et al. (2008) introduced three types of parallel MCTS, leaf parallelism, tree parallelism, and root parallelism, which balance exploration and exploitation through local mutex locks and the addition of virtual losses. Liu et al. (2020) implemented a master–worker parallel architecture and proposed a novel parallel MCTS algorithm, achieving notable results in games such as Joy City and Atari. Huang et al. (2022) developed a graphics processing unit (GPU)-based large-scale simulator for parallel MCTS algorithms to address robotic planning tasks. Meng et al. (2022) employed a scalable CPU field programmable gate array (CPU-FPGA) system for parallel MCTS, significantly enhancing the speed of in-tree operations. Yang et al. (2021) presented a large-scale parallel MCTS algorithm, applying distributed MCTS to real-world and non-game-related problems.

Parallel MCTS effectively improves the search speed of the game tree in both gaming and other application scenarios. Consequently, this paper incorporates parallel MCTS technology into the proposed model.

3 Methodology

Existing literature typically adopts different strategies for the various stages of Tibetan Jiu chess, such as using DRL models for the layout phase and human knowledge models for the battle phase. However, no studies have applied DRL throughout all stages. Unlike existing methods, JFA consists of two models, SLM and HBM, which are independently trained before being integrated, as shown in Fig. 1. Both models guide each other through deep neural networks and parallel MCTS to learn chess strategies. The strategy output P and value output V of the neural networks are used to evaluate nodes in the parallel MCTS, and the decision result (s, π) from MCTS, together with the rewards r provided by an auxiliary agent based on human knowledge, form the self-play training data for the neural networks. The auxiliary agent assists the SLM by simulating decisions in the battle phase and providing rewards r based on the square capturing state, influencing the layout phase. Similarly, the auxiliary agent works with the HBM, simulating layout phase decisions and providing rewards based on the actual game outcome. Human knowledge-based pruning methods are applied to optimize the action space and improve decision-making efficiency.

3.1 Parallel MCTS

Both SLM and HBM guide each other through deep neural networks and parallel MCTS to learn Tibetan Jiu chess strategies. The combination of deep neural networks and MCTS is a commonly used method in game theory. To improve self-play efficiency, various parallel MCTS methods have been proposed. In Tibetan Jiu chess, Wang YJ et al. (2022) attempted to optimize the traditional UCT algorithm using parallel multiprocessing, focusing on the leaf node simulations. Inspired by WU-UCT (Liu et al., 2020), we adopt multi-threaded parallel

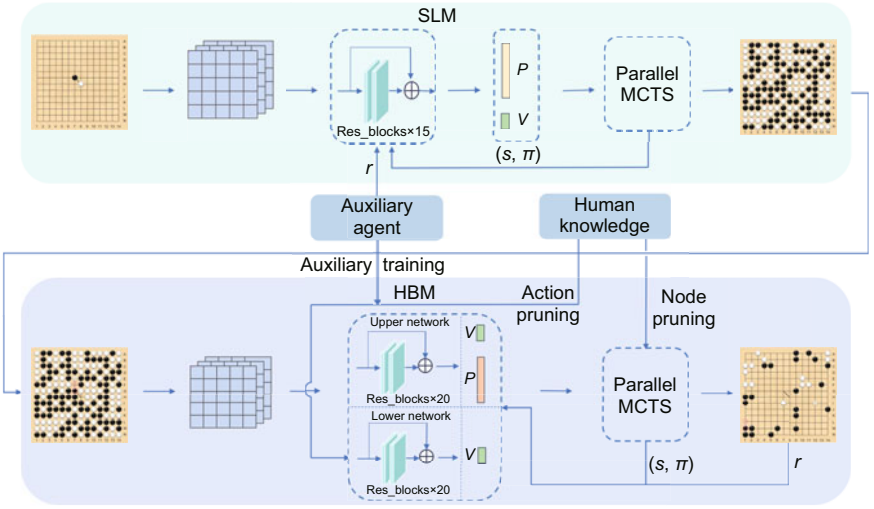

Fig. 1 JFA structure diagram. Res_blocks denotes the residual blocks

techniques to optimize the tree search.

MCTS is a model-based RL algorithm that selects the optimal action at each decision point (see Section 3.1 in the supplementary materials). In our designed parallel MCTS, we use Eq. (1) to select the action (edge) for each node based on the tree policy:

$$a_{\text{upper}} = \arg\max_{a\in\mathcal{A}(s)} \left(Q(s,a) + cP(s,a)\right) \cdot \frac{\sqrt{N(s)+O(s)}}{1+N(s,a)+O(s,a)}, \tag{1}$$

where $Q(s,a)$ represents the average value of selecting action a at state s, and $P(s,a)$ is the probability of action a derived from the policy head of the deep neural network. c represents the exploration parameter, $N(s)$ denotes the visit count of node s, and $N(s,a)$ represents the visit count of action a at state s. $O(s)$ counts the number of rollouts initiated but not yet completed. $O(s,a)$ denotes the number of initiated but not yet completed visits where action a is selected under node s. $\mathcal{A}(s)$ denotes the set of all possible actions that can be taken at state s.

We ensure that the parallel search process is efficient and maintains data consistency by using local locks to prevent multiple threads from simultaneously modifying shared data. In particular, we use incomplete and complete updates to track $N(s)$ and $O(s)$ along the traversed path. The incomplete update is performed before the simulation task starts, allowing the updated statistics to be instantly available globally, as shown in Eq. (2):

$$O(s) \leftarrow O(s) + 1. \tag{2}$$

The complete update is performed after the simulation task completes, ensuring that the updated statistics are instantly available globally, as shown in Eqs. (3) and (4):

$$O(s) \leftarrow O(s) - 1, \tag{3}$$

$$N(s) \leftarrow N(s) + 1. \tag{4}$$

In Eq. (1), $Q(s,a)$ is calculated as shown in Eq. (5), where v_i represents the reward obtained on the i^{th} visit to state s while taking action a, and $N(s,a)$ denotes the number of times action a has been taken in state s:

$$Q(s,a) = \frac{\sum_{i=1}^{N(s,a)} v_i}{N(s,a)}. \tag{5}$$

3.2 Decision process of HBM based on hierarchical networks

In Tibetan Jiu chess, the action categories during the battle phase include movement, jump capturing, and square capturing, resulting in a large action space. In previous studies, a single unified model was typically used to handle decision-making for the entire battle phase. However, this approach often struggles with efficiency and accuracy when faced with such a large action space. To more accurately simulate decision-making in Tibetan Jiu chess, we design a hierarchical network for the HBM. The upper network is responsible for movement and jump capturing, while the lower network is responsible for square capturing. By this design, the action space

is significantly diminished. Specifically, each movement or jump capturing action could originally be associated with 19 307 possible square capturing moves, causing an explosive growth in the overall action space. The hierarchical network decomposes these three action types into two independent subtasks, thereby reducing the computational complexity for each network. The upper network only needs to handle 10 388 movement and jump capturing actions, while the lower network processes the 19 307 possible square capturing moves. This hierarchical design allows the model to focus on a smaller set of actions in each subtask, effectively improving decision efficiency and accuracy while alleviating computational burden.

As shown in Fig. 2, at the current state S_t, HBM performs multiple parallel MCTS simulations to select an action a_t, and then updates the game to the next state S_{t+1}. In each parallel MCTS simulation, when the root node is created, the upper network is first called to guide the expansion of the root node. During the selection phase, the most valuable move a_{upper} is chosen according to Eq. (1). In the expansion and simulation phases, after executing a_{upper}, an intermediate state s_{t_1} is obtained. The next steps depend on whether executing a_{upper} results in the formation of a square, a special shape in the game.

If executing a_{upper} results in a square formation, the lower network is further called for more precise decision-making. Based on the decision information from a_{upper}, the lower network first uses pruning methods to trim the set of valid capturing actions, thereby reducing the computational cost of the lower network. Then, the action a_{lower} that maximizes the Q-value for the intermediate state s_{t_1} is selected, as shown in Eq. (6):

$$a_{\text{lower}} = \arg \max_{a \in A_{\text{eat}}(s_{t_1}, a_{\text{upper}})} Q(s, a_{\text{upper}}, a), \quad (6)$$

where $A_{\text{eat}}(s_{t_1}, a_{\text{upper}})$ represents the set of valid capturing actions that have been pruned after considering the upper network's action a_{upper} in state s_{t_1}. After executing a_{lower}, the state s_{t_2} is obtained. At state s_{t_2}, the upper network is called again, and the policy P and value V output by the network are used to guide node expansion and evaluation.

If executing a_{upper} does not form a square, the lower network is not called, and the state s_{t_1} is treated as the final state. In this case, the upper network is called at state s_{t_1}, and the policy P and value V are used to guide node expansion and simulation. After expanding the node, pruning methods are applied to eliminate low-value branches, improving the efficiency of the game tree search. During the backpropagation phase of the parallel MCTS, the evaluation values are propagated upward along the tree to update the evaluations of the relevant nodes. The details of the pruning methods are described in Section 3.5.

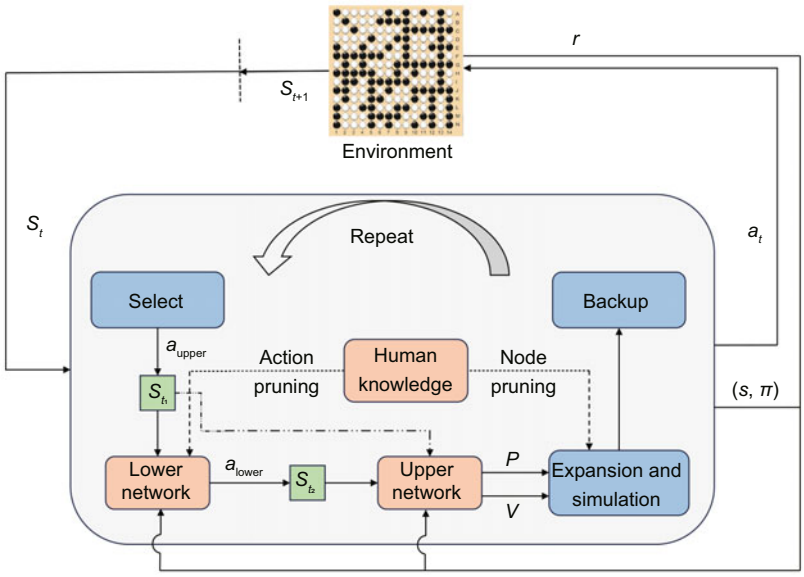

Fig. 2 Hierarchical network guiding the parallel MCTS decision-making process

3.3 Network structures of SLM and HBM

3.3.1 Network structure of SLM and upper network structure of HBM

The structure of the SLM network and the upper network structure of HBM are similar, differing in both the number of residual blocks and the size of their policy output heads. Specifically, SLM consists of 15 residual blocks, while the upper network of HBM contains 20 residual blocks. Taking SLM as an example, we describe its network structure. As shown in Fig. 3, the network input first passes through a convolutional block, then through a residual tower composed of multiple residual blocks, and finally outputs the strategy and value through the policy head and value head, respectively. Each convolutional block contains a 3×3 convolutional layer, a batch normalization layer, and a rectified linear unit (ReLU) activation function. Each residual block consists of two convolutional blocks. For the detailed definitions of the network input tensors, output dimensions, and loss function, see Section 3.2 in the supplementary materials.

3.3.2 Lower network structure of HBM

The lower network of HBM, like the upper network, consists of 20 residual blocks and is a dual-input, single-output DQN. As shown in Fig. 4, the lower network has two input heads and one value output head. One input head receives the chessboard information tensor and extracts deep features through the residual blocks, while the other input head receives the index of the action from the upper network, generating feature representations through an embedding layer. After concatenating both sets of features, they pass through convolutional and fully connected (FC) layers, ultimately outputting a scalar value, which is used to select the lower network's action. For the detailed definitions of network input tensors, output dimensions, and loss function, see Section 3.3 in the supplementary materials.

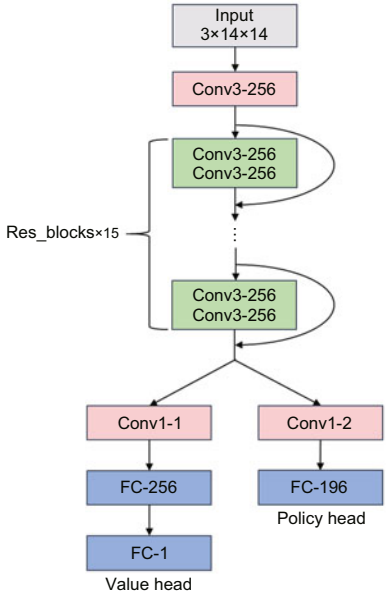

Fig. 3 Network structure of SLM

3.4 Reward design for SLM based on an auxiliary agent

After the layout phase is completed, the game remains ongoing so that the performance of the layout cannot be evaluated based on the game outcome. Other researchers have evaluated the layout by using shape evaluations and assigning rewards accordingly. However, this evaluation is inaccurate. To improve the accuracy of the evaluation, we design a reward system for the SLM based on an auxiliary agent.

The auxiliary agent is a Tibetan Jiu chess AI developed based on human knowledge, with a level roughly equivalent to that of a human amateur 2-dan player. The auxiliary agent works with the SLM to simulate the battle phase and provides rewards

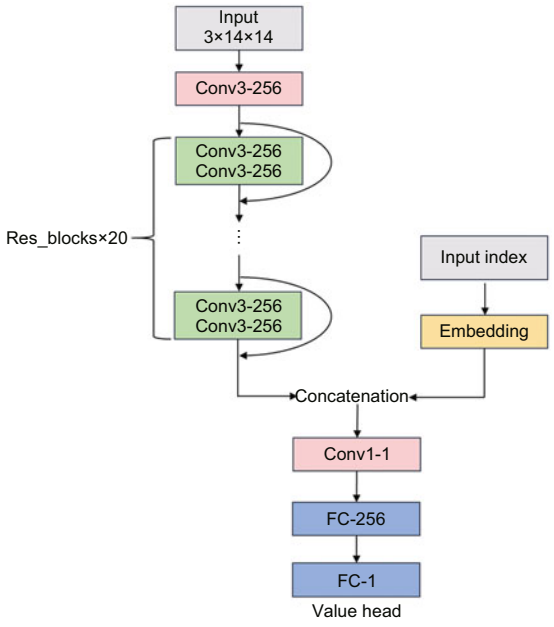

Fig. 4 Network structure of the lower network in HBM

to the layout phase based on the square formation. The first player to form a square during the game receives a +1 reward, while the opponent receives a −1 reward.

This reward design is reasonable. After training SLM using supervised learning with human player data as the training set, SLM learns basic layout strategies. However, it is unable to consider the entire game, particularly the crucial move "square formation," which is extremely advantageous in the battle phase. Therefore, by using the auxiliary agent to simulate the game and providing rewards based on whether the square formation is achieved, SLM can be effectively trained to not only consider the layout phase but also develop an intuitive understanding of the entire game, improving its decision-making ability. The generalizability of the handcrafted SLM reward design is detailed in Section 3.4 in the supplementary materials.

3.5 Knowledge-based pruning methods for HBM

During the battle phase, due to the large action space, HBM needs to process a vast number of legal actions, resulting in a significant increase in the search time and a decrease in the quality of self-play generated data. Pruning can accelerate data generation and improve decision-making quality. In previous studies, pruning methods for the MCTS expansion step have been proposed, which reduce the search space by eliminating moves that do not form a square and single-step jump actions. However, we propose a more refined pruning strategy that integrates human knowledge. This strategy not only is used to prune unimportant nodes during the parallel MCTS expansion phase, but also effectively reduces the number of actions that need to be evaluated by the lower network, thus improving both decision-making efficiency and accuracy.

3.5.1 Branch pruning in the expansion phase of parallel MCTS

In parallel MCTS, after a node is expanded, a pruning algorithm is applied to eliminate low-value branches of that node. Specifically, we categorize the legal actions into four levels, ranging from the first level (highest priority) to the fourth level (lowest priority), as shown in Table 1. If higher-level actions are available, all lower-level actions are pruned (see Section 3.5 in the supplementary materials).

3.5.2 Pruning of candidate actions in the HBM lower network

During the battle phase, the maximum number of square capturing actions is 4656 ($\binom{97}{2}$). If all legal actions are input into the DQN, parallel MCTS would require 4656 calls to the DQN each time a node is expanded, which severely affects the execution speed of the parallel MCTS. To address this, we design a pruning method based on human knowledge to prune square capturing actions, optimizing the action space.

For the current state s, after executing the square capturing action a, the next state s' is reached. At this point, both the player's and the opponent's evaluations of the state s' are performed. The specific evaluation process for each side is as follows:

First, all legal actions b's in state s' are iterated over, and the action b is scored based on Table 1. The resulting score is stored in the data dictionary (score_dict), with the key being the end position, storing the maximum score ($\text{Score}_{\max}$) at that position. If multiple legal actions have the same end position, the new score (new_score) is compared with the stored value score_dict[end_position], and the maximum of the two is selected, as shown in Eq. (7):

$$\text{Score}_{\max} = \max(\text{score\_dict}[\text{end\_position}], \text{new\_score}). \tag{7}$$

Table 1 Action classification and description

Level	Description	Pruning principle
Level 1	Movement of the Dalian	If present, prune actions from lower levels
Level 2	Movement of the square	If present, prune actions from levels 3 and 4
Level 3	Jump capturing and opening a square	If present, prune actions from level 4
Level 4	Invalid legal movements	No pruning

Then, the total score for a side at state s' is calculated using Eq. (8):

$$f(s', \text{role}) = \sum_{\text{end\_position} \in \text{score\_dict}} \text{score\_dict}[\text{end\_position}]. \tag{8}$$

Finally, the score difference between the opponent and the player is calculated using Eq. (9):

$$F(s') = 1.5 \cdot f(s', o) - f(s', p_{\text{l}}), \tag{9}$$

where $F(s')$ represents the score for the square capturing action a. Here, o represents the opponent and p_{l} represents the player. The opponent's score is scaled by 1.5 to reduce the potential score in the next round.

In Table 2, we evaluate each legal action by assessing the number of "squares" formed at both the start and end positions, as well as whether the formation created by the action can be broken by the opponent to assign a score to each action.

Table 2 Score table of legal movements

N_{s}	N_{e}	Destroyable	Score
0	1	Yes	2
1 or 2	1	Yes	4.5
0	1	No	2.5
1 or 2	1	No	9.5
0	2	Yes	3
1 or 2	2	Yes	4.5
0	2	No	4
1 or 2	2	No	9.5

N_{s} is the number of squares at the starting point, N_{e} is the number of squares at the endpoint, and destroyable denotes whether it can be destroyed

We adopt different pruning strategies for square capturing one piece and two pieces. For capturing one piece, actions are divided into high-value and low-value sets based on the square capturing action scoring function $F(s')$. If high-value actions exist, low-value actions are pruned. For capturing two pieces, the occurrence is less frequent, but once it happens, even with the above strategy, there may still be many possible action combinations. To address this, we directly sort the actions according to the scoring function $F(s, a)$ and select the top 20 actions with the highest scores. The specific process is detailed in Section 3.6 in the supplementary materials.

3.6 Comparison with AlphaGo and AlphaZero

AlphaGo is trained on large-scale human game records using a supervised policy network to obtain high-quality move priors. It subsequently introduces a fast policy network to support efficient simulations. Through self-play, it optimizes a reinforcement policy network that shifts the learning objective from imitating human play to maximizing winning probability. A value network is then trained to predict the game outcome from positions generated by the reinforcement policy's self-play. In MCTS, AlphaGo incorporates the policy priors into tree expansion and combines the outputs of the value network with fast simulation results when evaluating leaf nodes with the reward signal derived solely from the final game outcome.

In contrast, AlphaZero simplifies the architecture into a single residual network that jointly outputs both policy and value, relying entirely on self-play and MCTS-driven iterative optimization. This approach offers stronger generality but demands extremely high computational resources and large amounts of training data, which makes direct application challenging in scenarios with complex rules or constrained computational budgets. Its reward design likewise adopts the terminal game outcome as the sole signal.

Distinct from both, JFA introduces structural modifications and innovations tailored to the unique characteristics of Tibetan Jiu chess. Specifically, since the game consists of two distinct phases—the layout phase and the battle phase—JFA departs from the single-stage modeling paradigm of AlphaGo and AlphaZero by employing a two-stage architecture: the SLM dedicated to optimizing layout decisions and the HBM specialized for battle strategies. Each network adopts a design similar to AlphaZero, producing both policy and value outputs within a single framework, but further incorporates a hierarchical structure to decouple different action types, thereby enhancing the expressive capacity of the policy representation.

In terms of training methodology, JFA integrates AlphaGo's supervised initialization with AlphaZero's self-play mechanism: It is first pretrained on a limited set of high-quality human records to stabilize early learning, and then

transitions into self-play to achieve iterative policy refinement, thus improving efficiency while reducing reliance on extensive computational resources. Regarding reward design, JFA introduces auxiliary agents to provide additional signal sources, rather than relying exclusively on terminal outcomes. Moreover, in parallel MCTS, JFA abandons AlphaZero's virtual loss mechanism, and instead adopts the WU–UCT method, which employs latent variables to coordinate parallel exploration. Combined with human knowledge-based pruning strategies during tree search, this significantly enhances both search efficiency and decision quality.

4 Experimental results and analysis

The experiments are divided into three parts. The first part verifies the performance of the SLM, specifically assessing the impact of parallel MCTS on decision-making time and evaluating the decision-making capability of the SLM. The second part validates the decision-making capability of the HBM. The third part evaluates the overall performance of the JFA agent by testing the cooperative interaction between the SLM and HBM.

4.1 Dataset and model training

The model training is conducted in two stages. The first stage employs a preprocessed dataset for supervised learning, while the second stage uses the data collected through self-play for RL training.

4.1.1 Dataset

The dataset used for supervised learning originates from 1700 valid game records collected via the research group's Tibetan Jiu chess strategy mini-program, which are cleaned to remove incomplete or illegal games and augmented by applying a 180° board rotation (see Section 4.1 in the supplementary materials).

4.1.2 Model self-play training

In the training of SLM and HBM, an independent training strategy is adopted with the support of an auxiliary agent to enhance the effectiveness and efficiency of the networks. During SLM's self-play training, the auxiliary agent primarily provides reward signals for the layout phase. When training the upper and lower networks of HBM simultaneously, changes in the lower strategy can affect the effectiveness of the upper strategy, and vice versa. To achieve independent training of the upper and lower networks within HBM, the auxiliary agent is introduced as follows: During the upper network's self-play, the agent is responsible for square capturing, while during the lower network's self-play, it is responsible for piece movements (see Section 4.2 in the supplementary materials).

4.2 Baseline models

In the experiments, we use two baseline models. The first baseline model is ShapeAgent for the layout phase. This agent is replicated based on the methodology by Wang SY (2023) and adapted to a 14×14 chessboard. ShapeAgent employs the AlphaZero algorithm during the layout phase and uses an expert knowledge model during the battle phase. After the layout phase ends, it receives rewards based on a chess formation evaluation function, making it the highest-performing Tibetan Jiu chess layout model reported in the literature to date.

The second baseline model is ExpertAgent, also replicated from Wang SY (2023). This model combines expert knowledge with greedy strategies, making it a top-tier model for the entire game phase of Tibetan Jiu chess.

4.3 SLM experiments and result analysis

4.3.1 Impact of the parallel MCTS on decision-making time

The performance of parallel MCTS is evaluated by measuring the average decision-making time per move of the agent. SLM using parallel MCTS is compared against SLM using standard MCTS, with parallel MCTS using 16 threads. Table 3 presents a comparison of the average decision-making time per move under different numbers of MCTS simulations.

As shown in Table 3, the reduction in decision-making time achieved by using parallel MCTS is highly significant, validating the crucial role of parallel MCTS in enhancing the decision-making efficiency of the model. The substantial decrease in the average decision-making time per move also plays a vital role in accelerating the self-play speed of the model, thereby improving the overall efficiency of self-play training.

Table 3 Comparison of average decision-making time per move for different numbers of MCTS simulations

Simulation number	Decision time per move (s)		Reduction factor
	Parallel MCTS	MCTS	
400	0.43	6.17	14.35
800	0.79	12.63	15.99
1200	1.17	18.26	15.61
1600	1.59	24.32	15.30
2000	1.92	30.43	15.85

4.3.2 Experiment and analysis of SLM's decision-making capability

ShapeAgent plays 100 games against SLM, with SLM as white in 50 games and black in the other 50 games. After the layout phase, ShapeAgent executes the battle phase for both sides. An MCTS with 400 simulations was used to select each move. The game results are illustrated in Fig. 5.

As shown in Fig. 5, SLM achieves an overall win rate of 74%. Specifically, SLM attains an 84% win rate when playing as white and a 64% win rate when playing as black. These results indicate that SLM possesses superior decision-making capabilities. Compared with the traditional position evaluation function reward mechanism used by ShapeAgent, SLM introduces an auxiliary agent-based reward mechanism, which delivers more precise reward signals, enhances the model's learning effectiveness, and consequently contributes to its superior win rate performance. In terms of average decision-making time per move, SLM demonstrates a remarkable advantage, achieving a speed increase of approximately 47 times. This improvement is attributed to two main factors: First, parallel MCTS significantly accelerates the decision-making process, and second, the implementation language plays a role—SLM is implemented in C++, whereas ShapeAgent is implemented in Python.

4.3.3 SLM wins the championship of the 2024 National Computer Games Tournament in China

The SLM participated in the 18$^{\text{th}}$ China University Student Machine Game Competition and the China Machine Game Championship, held in Xining, Qinghai Province, China, in 2024. Competing in the Tibetan Jiu chess category, SLM faces all other participating AI agents and achieves an undefeated record, securing the national championship.

The SLM demonstrates excellent layout capabilities, successfully achieving the objective of enabling the agent to form squares first during the battle phase (see Section 4.3 in the supplementary materials). This strategic advantage enables SLM to win the 2024 Tibetan Jiu chess category. The outstanding performance of SLM is attributed to its reward design, which is based on the auxiliary agent's evaluation of early square formation during the battle phase. This outcome validates the effectiveness of the proposed reward design method.

4.4 Experimental analysis of HBM's decision-making capability

We conducted 100 games between HBM and the baseline model, ExpertAgent (in the battle phase), with both sides using ExpertAgent in the layout phase and their respective models in the battle phase. HBM was played as black and white in 50 games each. An MCTS with 200 simulations was used to select each move, and the game outcomes are presented in Fig. 6.

As depicted in Fig. 6, HBM achieves a win rate of 73% when playing as white and 67% when playing as black, resulting in an overall win rate of 70%. These results indicate that HBM demonstrates strong flexibility and adaptability in handling complex game scenarios. It is capable of accurately analyzing intricate positions and adjusting its strategies to make superior decisions. This validates the effectiveness of HBM in game state analysis and decision-making within Tibetan Jiu chess.

4.5 Overall performance of the JFA agent

4.5.1 Performance of the combined SLM and HBM

The JFA agent employs the SLM during the layout phase and the HBM during the battle phase. We conducted 100 games between JFA and the baseline model, ExpertAgent, with JFA playing as black and white in 50 games each. SLM used an MCTS with 900 simulations per move, while HBM employed an MCTS with 200 simulations per move. The game outcomes are detailed in Table 4.

To better compare the synergistic effects of the SLM and HBM, Table 4 also includes data from 100 games where the SLM and HBM are individually matched against other models.

As shown in Table 4, JFA achieves an impressive

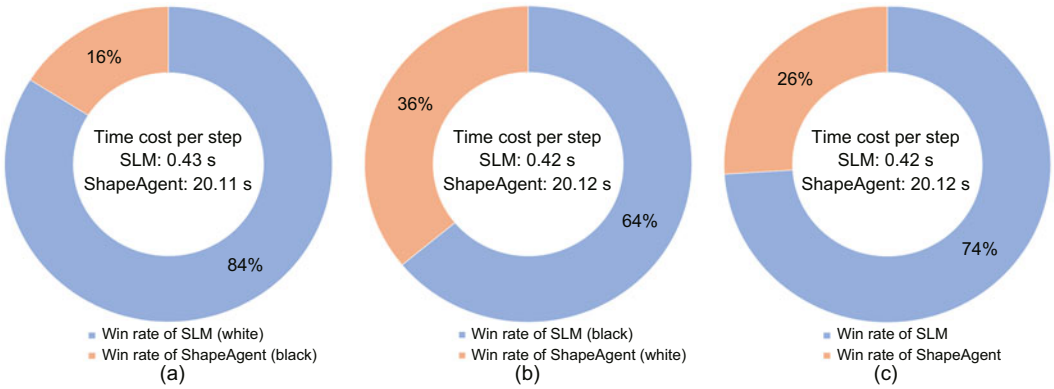

Fig. 5 Results of 100 games for SLM vs. ShapeAgent (layout phase): (a) win rate of SLM with white; (b) win rate of SLM with black; (c) win rate of SLM

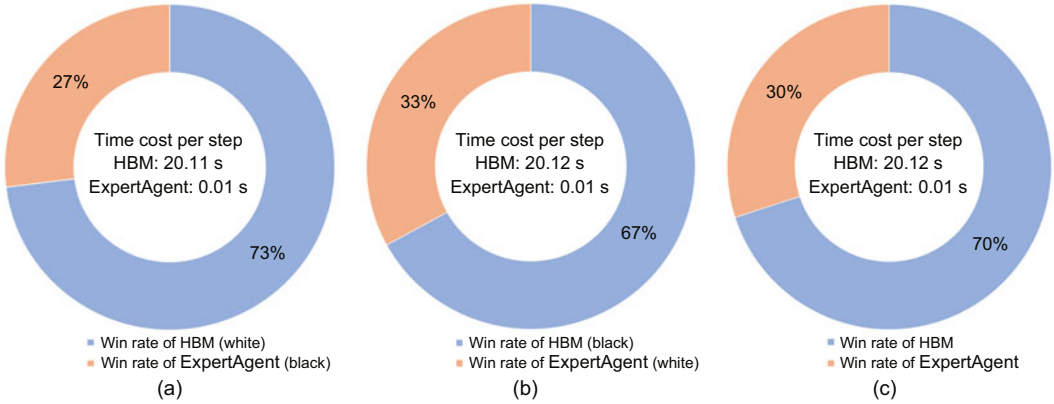

Fig. 6 Results of 100 games for HBM vs. ExpertAgent (battle phase): (a) win rate of HBM with white; (b) win rate of HBM with black; (c) win rate of HBM

Table 4 Match results between JFA and baseline models

Agent	Win	Fail	Win rate (%)	Standard error	95% confidence interval
JFA vs. ExpertAgent	81	19	81	0.0392	[73.31%, 88.69%]
SLM vs. ShapeAgent	74	26	74	0.0439	[65.40%, 82.60%]
HBM vs. ExpertAgent	70	30	70	0.0458	[61.02%, 78.98%]

win rate of 81% against the baseline model, outperforming the win rates of SLM at 74% and HBM at 70%. These experimental results demonstrate that the collaboration between the two models enhances strategic generation capabilities when dealing with complex game states. This corroborates the proposed approach of an independently trained two-stage model based on hierarchical neural networks and human knowledge, effectively elevating the playing strength of the Tibetan Jiu chess AI agent.

4.5.2 Pruning ablation experiment

This experiment aims to evaluate the necessity of the two pruning algorithms in JFA and their impact on model performance. In JFA, pruning is primarily used to enhance decision quality and speed. We therefore perform ablations of each pruning strategy and measure JFA's win rate against ExpertAgent and HBM's decision time, while keeping all other settings constant. As shown in Table 5, we report results under four pruning configurations.

As shown in Table 5, when both pruning layers are enabled, the model achieves a win rate of 81% with an average decision time of 20.12 s, demonstrating a favorable balance. Expansion-phase pruning reduces ineffective nodes during MCTS expansion, while lower-level move pruning optimizes the number of candidate moves, thereby improving

Table 5 JFA vs. ExpertAgent under different pruning configurations

Pruning configuration	Win rate (%)	Decision time (s) (for HBM)
Both pruning configurations enabled	81	20.12
Without expansion-phase pruning only	51↓	6.47↓
Without lower-level move pruning only	64↓	41.18↑
Both pruning configurations removed	<20↓	2.20↓

computational efficiency and win rate. When expansion-phase pruning is removed, decision time decreases substantially to 6.47 s, but the win rate drops sharply to 51%. This indicates that node pruning is critical for decision quality—without it, the search space balloons, leading to reduced win rates. The shorter decision time reflects the overhead introduced by the pruning algorithm. When lower-level move pruning is disabled, the win rate falls to 64%, while decision time increases to 41.18 s, showing that lower-level pruning plays a key role in computational efficiency. Without it, the number of candidate moves grows, significantly prolonging computation. Finally, with both pruning strategies removed, decision time is only 2.20 s, but the win rate drops below 20%, indicating that an unpruned decision space is too large for the model to make effective decisions. Although pruning incurs some time overhead, its ability to substantially reduce computational burden and optimize the decision space far outweighs this cost. Therefore, pruning is indispensable in our model.

4.5.3 Demonstration of JFA's advanced chess playing capabilities

Through self-play RL, JFA has achieved a high level of strategic intelligence. It has not only mastered the fundamental aspects of board layout, piece movement, and basic rules, but also learned advanced formations and offensive and defensive strategies (see Section 4.4 in the supplementary materials).

4.6 Discussion

Based on hierarchical neural networks and human knowledge guidance, the two-stage model architecture proposed in this paper demonstrates exceptional performance in enhancing the AI capabilities of Tibetan Jiu chess while reducing decision-making time. Although JFA is specifically tailored for Tibetan Jiu chess, its core design principles offer significant transfer potential to other complex tasks. For example, JFA's staged decomposition and hierarchical action strategy can be applied to games with well-defined phase structures or large heterogeneous action spaces—such as the macro and micro level operations in real-time strategy (RTS) games. Moreover, the parallel MCTS and pruning strategies employed by JFA further enhance the framework's generality, enabling it to address a variety of decision-making problems, including robot path planning and resource allocation systems. When adapting JFA to a new domain, one must redesign key elements—such as phase segmentation, action type identification, the form of auxiliary agents, and pruning strategies—to suit the specifics of the target task. However, JFA exhibits reduced robustness in extreme or rare game scenarios, and incurs slower decision-making in its hierarchical model (see Section 4.5 in the supplementary materials).

5 Conclusions

To enhance the decision-making capabilities of DRL models for Tibetan Jiu chess within the constraints of limited hardware resources, we propose JFA, a model that combines the SLM designed specifically for the layout phase with the HBM designed for the battle phase to make decisions. Both models employ similar network architectures and use parallel MCTS methods. These models are trained independently through self-play. Additionally, we introduce a human knowledge-based auxiliary agent to assist in the independent training of the SLM and HBM. The HBM incorporates hierarchical neural networks, and two knowledge-based methods are proposed to prune sub-optimal branches in the parallel MCTS and capturing actions in the lower network.

When competing against a layout model based on the AlphaZero method, SLM achieves a 74% win rate while significantly reducing decision-making time. The JFA, using both SLM and HBM, reaches

a performance level comparable to that of a human amateur 4-dan player, making it the best-performing Tibetan Jiu chess AI to date. Additionally, SLM secured the championship at the 2024 China National Computer Game Tournament, further validating the effectiveness of our dual-model approach in enhancing the AI capabilities of Tibetan Jiu chess and reducing resource consumption.

Future research may focus on further optimizing the two-stage model and exploring its application potential in other complex strategy games. By integrating more human knowledge and advanced deep learning techniques, ongoing improvements could elevate both the intelligence and computational efficiency of AI systems.

Contributors

Xiali LI designed the research. Xiaoyu FAN designed the system, conducted the experiments, and collected and processed the data. Xiali LI and Xiaoyu FAN drafted the paper. Junzhi YU helped organize the paper. Zhicheng DONG, Xianmu CAIRANG, and Ping LAN helped revise the paper. Xiali LI and Junzhi YU revised and finalized the paper.

Conflict of interest

All the authors declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from Xiali LI (E-mail: lixiali@muc.edu.cn) upon reasonable request.

References

Brown N, Sandholm T, 2017. Safe and nested subgame solving for imperfect-information games. Proc 31st Int Conf on Neural Information Processing Systems, p.689-699.

Brown N, Sandholm T, 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418-424. https://doi.org/10.1126/science.aao1733

Chaslot GMB, Winands MHM, van den Herik HJ, 2008. Parallel Monte-Carlo tree search. 6th Int Conf on Computers and Games, p.60-71. https://doi.org/10.1007/978-3-540-87608-3_6

Chu SJ, 2024. APSN: adaptive prediction sample network in deep Q learning. Proc 3rd Int Conf on Algorithms, Microchips, and Network Applications, p.461-465. https://doi.org/10.1117/12.3031933

Haarnoja T, Zhou A, Abbeel P, et al., 2018. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. Proc 35th Int Conf on Machine Learning, p.1861-1870.

Hessel M, Modayil J, Van Hasselt H, et al., 2018. Rainbow: combining improvements in deep reinforcement learning. 32nd AAAI Conf on Artificial Intelligence, p.3215-3222. https://doi.org/10.1609/aaai.v32i1.11796

Huang BC, Boularias A, Yu JJ, 2022. Parallel Monte Carlo tree search with batched rigid-body simulations for speeding up long-horizon episodic robot planning. IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.1153-1160. https://doi.org/10.1109/IROS47612.2022.9981962

Itaya H, Hirakawa T, Yamashita T, et al., 2024. Mask-attention A3C: visual explanation of action-state value in deep reinforcement learning. *IEEE Access*, 12:86553-86571. https://doi.org/10.1109/ACCESS.2024.3416179

Li JJ, Koyamada S, Ye QW, et al., 2020. Suphx: mastering Mahjong with deep reinforcement learning. https://arxiv.org/abs/2003.13590

Li XL, Wang S, Lv ZY, et al., 2018. Strategy research based on chess shapes for Tibetan Jiu computer game. *ICGA J*, 40(3):318-328. https://doi.org/10.3233/ICG-180058

Li XL, Lv ZY, Wu LC, et al., 2020. Hybrid online and offline reinforcement learning for Tibetan Jiu chess. *Complexity*, 2020(1):4708075. https://doi.org/10.1155/2020/4708075

Li XL, Chen YD, Zhang YY, et al., 2023. A phased game algorithm combining deep reinforcement learning and UCT for Tibetan Jiu chess. IEEE 47th Annual Computers, Software, and Applications Conf, p.390-395. https://doi.org/10.1109/COMPSAC57700.2023.00059

Li XL, Zhang YY, Wu LC, et al., 2024. TibetanGoTinyNet: a lightweight U-Net style network for zero learning of Tibetan Go. *Front Inform Technol Electron Eng*, 25(7):924-937. https://doi.org/10.1631/FITEE.2300493

Liang ZH, Li G, Gu RQ, et al., 2024. SampleViz: concept based sampling for policy refinement in deep reinforcement learning. IEEE 17th Pacific Visualization Conf, p.359-368. https://doi.org/10.1109/PacificVis60374.2024.00051

Liu AJ, Chen JS, Yu MZ, et al., 2020. Watch the unobserved: a simple approach to parallelizing Monte Carlo tree search. https://arxiv.org/abs/1810.11755

Meng Y, Kannan R, Prasanna V, 2022. Accelerating Monte-Carlo tree search on CPU-FPGA heterogeneous platform. 32nd Int Conf on Field-Programmable Logic and Applications, p.176-182. https://doi.org/10.1109/FPL57034.2022.00037

Mnih V, Kavukcuoglu K, Silver D, et al., 2013. Playing Atari with deep reinforcement learning. https://arxiv.org/abs/1312.5602

OpenAI, 2019. Dota 2 with large scale deep reinforcement learning. https://arxiv.org/abs/1912.06680

Perez D, Samothrakis S, Lucas S, 2014. Knowledge-based fast evolutionary MCTS for general video game playing. IEEE Conf on Computational Intelligence and Games, p.1-8. https://doi.org/10.1109/CIG.2014.6932868

Silver D, Huang A, Maddison CJ, et al., 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484-489. https://doi.org/10.1038/nature16961

Silver D, Schrittwieser J, Simonyan K, et al., 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354-359. https://doi.org/10.1038/nature24270

Silver D, Hubert T, Schrittwieser J, et al., 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140-1144. https://doi.org/10.1126/science.aar6404

Tian YD, Gong QC, Shang WL, et al., 2017. ELF: an extensive, lightweight and flexible research platform for real-time strategy games. Proc 31st Int Conf on Neural Information Processing Systems, p.2656-2666.

Vinyals O, Babuschkin I, Czarnecki WM, et al., 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350-354. https://doi.org/10.1038/s41586-019-1724-z

Wang SY, 2023. Research and Implementation of Computer Game Algorithm of Tibetan Jiu Chess. MS Thesis, Minzu University of China, Beijing, China (in Chinese).

Wang YJ, Liang K, Qiao JL, et al., 2022. The application of improved UCT combined with neural network in Tibetan Jiu chess. *Int J Wirel Mob Comput*, 23(1):22-32. https://doi.org/10.1504/IJWMC.2022.125530

Yang XF, Aasawat TK, Yoshizoe K, 2021. Practical massively parallel Monte-Carlo tree search applied to molecular design. https://arxiv.org/abs/2006.10504

Zhang XC, Liu L, Chen L, et al., 2021. An evaluation method for the computer game agent of the intangible heritage Tibetan Jiu chess item. *J Chongqing Univ Technol (Nat Sci)*, 35(12):119-126 (in Chinese). https://doi.org/10.3969/j.issn.1674-8425(z).2021.12.015

Zhang Z, Xu SL, Xia YY, et al., 2023. Two improved algorithms based on DQN. IEEE Int Conf on Signal Processing, Communications and Computing, p.1-4. https://doi.org/10.1109/ICSPCC59353.2023.10400213

List of supplementary materials