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

Supplementary materials for

Xiali LI, Xiaoyu FAN, Junzhi YU, Zhicheng DONG, Xianmu CAIRANG, Ping LAN, 2025. Jiu fusion artificial intelligence (JFA): a two-stage reinforcement learning model with hierarchical neural networks and human knowledge for Tibetan Jiu chess. *Front Inform Technol Electron Eng*, 26(10):1969-1983.
https://doi.org/10.1631/FITEE.2500287

1 Introduction to Tibetan Jiu chess

The battle phase includes actions such as movement, jump capture, and square capture, with square capture playing a pivotal role in determining the game's outcome. The complexity of actions in Tibetan Jiu chess far exceeds that of Go.

2 The rules of Tibetan Jiu chess

2.1 Layout Phase

The layout phase begins with an empty chessboard, with White making the first move. During the initial moves, both sides must place one piece on each diagonal line of the central grid squares, as illustrated in Figure S1. Subsequently, the players alternate placing pieces until all squares on the chessboard are occupied.

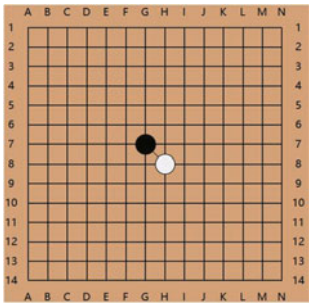


Fig. S1 Openings in the layout phase

2.2 Battle Phase

At the onset of the battle phase, both players remove their pieces from the diagonal lines on the chessboard. The player who placed the last piece during the layout phase (i.e., Black) takes the first turn to move, as depicted in Figure S2. In the battle phase, players alternate moves, selecting only one piece per turn. The available actions include movement and jump capture. When the endpoint of a move or jump capture results in the formation of a square (i.e., four of the player's pieces occupy the four corners of a grid cell), the player can perform a square capture, allowing them to remove an opponent's piece from any location on the board.

1. **Single-Step Movement**: A player's piece can move one grid space to an adjacent empty point. For example, in Figure S3(a), the Black piece at C3 can move one grid space to any surrounding empty point.

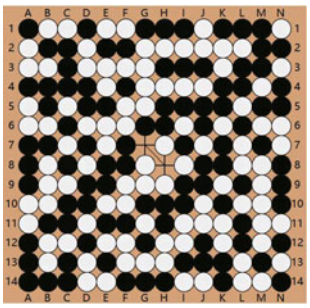
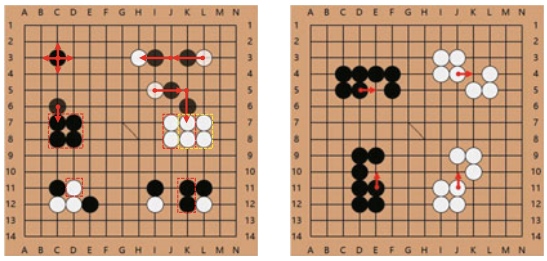


Fig. S2 Openings in the battle phase

2. **Jump Capture**: When a player's piece is adjacent to an opponent's piece along the same line, and there is an empty grid point beyond, the player can jump over the opponent's piece to land on the empty grid point, thereby capturing the opponent's piece. Jump captures can be chained as long as the conditions are met. For instance, in Figure S3(a), the White piece at L3 jumps left to J3, then from J3 jumps to H3, capturing the Black pieces at K3 and I3.

3. **Square Capture**: When the endpoint of a move or jump capture results in the formation of a square by the player's pieces, the player can capture any number of opponent's pieces equal to the number of squares formed. For example, in Figure S3(a), the Black piece at C6 moves downward to C7 along the red arrow, forming a square at C7 (red box), and subsequently captures the White piece at D11. Similarly, the White piece at I5 jumps right to K5, then from K5 jumps to K7, capturing the Black pieces at J5 and K6, thereby forming two squares at K7 (red and yellow boxes) and subsequently capturing the Black pieces at K11 and K12.



(a) Battle phase rules (b) Dalian formations

Fig. S3 Battle phase rules and Dalian formations

2.3 Classic Formation: Dalian

The Dalian is a classic formation composed of seven pieces by one side, existing in two variations: flat Dalian and diagonal Dalian, as shown in Figure S3(b). The Dalian formation can continuously form squares by moving a single piece back and forth, enabling the player to perform consecutive square captures. This formation is highly aggressive and strategically significant during the battle phase.

2.4 Judgment of Victory and Defeat

Considering the competition of the game consists of two stages, a player achieves victory when one of the following conditions is met:

1. During the battle phase, if a player forms two or more Dalian formations that the opponent is unable to disrupt, the player is declared the winner.

2. During the battle phase, if the opponent's remaining pieces are reduced to 14 or fewer, the player is declared the winner.

3 Supplementary Methods

3.1 Introduction to Monte Carlo Tree Search

Specifically, it uses an MDP model to construct a search tree to determine the optimal action at each step. Each node s represents a state-action pair (s, a), and each edge starting from s represents an action that can be taken in that state. MCTS performs four sequential steps: selection, expansion, simulation, and backpropagation. The MCTS algorithm is inherently sequential: each selection step in a new rollout requires the completion of previous rollouts in order to update the statistics for the UCT tree policy.

3.2 Network structure of SLM and upper network structure of HBM

A. Network Input

The input to the network is a tensor of dimensions $3 \times 14 \times 14$. The first 14×14 slice represents the current state of the chessboard, where 0 indicates an empty position, 1 denotes a black piece, and 2 signifies a white piece. The second and third slices represent the current player: if the current player is black, the second slice is entirely filled with 1; otherwise, the third slice is entirely filled with 1.

B. Network Output

The network features two output heads: the policy head and the value head. The policy head of SLM outputs a 196-dimensional vector through a fully connected layer, while the policy head of the upper network in HBM outputs a 10,388-dimensional vector. Both vectors represent the probabilities of various actions.The value head outputs a scalar within the range $[-1, 1]$, indicating the probability of the current player winning from the current state.

C. Loss Functions

The network utilizes an unweighted sum of the losses from its two output heads (see Equation (1)). The primary objective is to minimize the discrepancy between the actual game outcome z and the value head output v of the model, while simultaneously maximizing the similarity between the true policy π and the model-predicted action probability distribution p. z represents the game outcome from the perspective of the current player, which is derived from the final reward r, where $r = 1$ for a win and $r = -1$ for a loss. For SLM's upper network, the reward r is generated based on the auxiliary agent. On the other hand, for HBM, the reward r is directly determined by the actual game outcome, reflecting the win or loss of the game. An L_2 regularization term c is included to prevent overfitting.

$$\text{loss} = (z - v)^2 - \pi^T \log p + c||\theta||^2 \tag{1}$$

3.3 Lower Network Structure of HBM

A. Network Input

The lower network is a dual-input Deep Q-Network comprising a tensor of size $5 \times 14 \times 14$ and a candidate action index. Unlike the upper network, the lower network includes two additional 14×14 slices: one slice represents the positions of the opponent's pieces, and the other slice indicates the number of squares formed by the actions selected by the upper network. This configuration enables the network to process both the overall board state and specific action-related information.

B. Network Output

The lower network of HBM outputs a scalar within the range $[-1, 1]$, representing the probability of the current player winning from the given state. This output facilitates the selection of precise actions during the battle phase by assessing the effectiveness of potential moves based on both the board state and the actions proposed by the upper network.

C. Loss Functions

The network utilizes an unweighted mean squared error (MSE) loss to minimize the discrepancy between the predicted value v_i and the actual game outcome z (see Equation (2)). Here, z is derived directly from the final reward r, where $r = 1$ for a win and $r = -1$ for a loss, based on the game outcome at the conclusion of the self-play session. M is the total number of samples.

$$loss = \frac{1}{M} \sum_{i=1}^{M} (v_i - z)^2 \tag{2}$$

3.4 Generalizability of the Handcrafted SLM Reward Mechanism

We fully acknowledge the limitations of manually crafted reward signals, especially their applicability across different tasks and domains. In our current framework, we employ hand-designed signals that prove effective for a complex, strategy-intensive game like Tibetan Jiu Chess, but we agree that broader generalization warrants further consideration. Our reward scheme is not only grounded in domain expertise but also tailored to the specific objectives of each game phase. For example, the layout and battle phases of Tibetan Jiu Chess entail distinct goals and strategic requirements, so rewards are phase-specific and adjusted according to the phase objectives. We believe that whenever a task inherently involves a multi-stage decision process, a similar reward design approach can be transferred and applied in other domains. Moreover, this phase-based, handcrafted reward methodology can serve as a useful reference for other complex decision tasks or staged decision problems. Many fields—such as multi-stage path planning or decision making in autonomous driving—also exhibit phased structures in which early-stage goals differ from later execution objectives. In such tasks, manually designed reward signals can be adjusted and optimized in line with evolving task goals, ensuring coherence and efficiency in learning. Although bespoke tuning of hand-crafted signals may be required for different applications, the underlying principle—namely, adjusting reward signals in accordance with phase-specific objectives—can be migrated and applied across a wide range of complex tasks.

3.5 Pseudocode for Branch Pruning in the Expansion Phase of Parallel MCTS

The pseudocode of the pruning algorithm is presented in Algorithm S1 S1.

3.6 Pseudocode for Pruning Candidate Actions in the HBM Lower Network

The specific process is detailed in Algorithm S2 S2.

4 Supplementary Experimental Results and Analysis

4.1 Dataset Preparation and Augmentation

The original dataset was in CSV format, containing over 2,000 game records. However, after cleaning the data to remove games with missing or illegal moves, 1,700 games remained. The CSV-formatted data was then converted to HDF5 format to facilitate model training. Additionally, data augmentation was performed on the SLM training dataset by rotating the chessboard 180° clockwise, effectively doubling the size of the dataset.

4.2 Self-Play Training Details

In the initialization phase of training, the SLM network and the upper network of HBM are each trained using 1,700 human game records through supervised learning to obtain the initial network models. Subsequently, reinforcement learning is conducted through self-play. Due to the insufficient number of

Algorithm S1 Action branch pruning strategy in parallel MCTS

```
Input: state (current game state), actions (set of actions before filtering)
Output: categorized_actions (set of filtered actions) PruneActionsstate, next_player
categorized_actions ← { "level1": [], "level2": [], "level3": [], "level4": [] }
for each action in actions do
    if IsLevel1Action(action, state) then
        categorized_actions["level1"].append(action)
    else if IsLevel2Action(action, state) then
        categorized_actions["level2"].append(action)
    else if IsLevel3Action(action, state) then
        categorized_actions["level3"].append(action)
    else
        categorized_actions["level4"].append(action)
    end if
end for
for level in ["level1", "level2", "level3", "level4"] do
    if categorized_actions[level] then return categorized_actions[level]
    end if
end for
```

Algorithm S2 Pruning for square capturing moves

Input: s (current situation), $actions$ (square capture action set), $max\_capture$ (maximum capture count)
Output: $pruned\_actions$ (filtered action set)
if $max\_capture = 1$ **then**
  $high\_value\_actions, low\_value\_actions \leftarrow \emptyset, \emptyset$
  for each $action \in actions$ **do**
    $score \leftarrow F(s')$ ▷ Compute score after executing $action$
    if $score < F(s)$ **then**
      $high\_value\_actions$.add($action$)
    else
      $low\_value\_actions$.add($action$)
    end if
  end for
  $pruned\_actions \leftarrow high\_value\_actions$ **if** $high\_value\_actions \neq \emptyset$ **else** $low\_value\_actions$
else
  $actions\_scores \leftarrow [(action, F(s')) \text{ for } action \in actions]$
  $pruned\_actions \leftarrow$ select top 20 lowest-score actions from $actions\_scores$
end if

square capture actions in the dataset, the HBM lower network begins self-play training from scratch using an ϵ-greedy strategy.

During each round of self-play training, the current optimal network guides the parallel MCTS, and moves are determined by sampling based on visit counts. After generating 2,500 self-play games, the model is trained once, and model checkpoints are saved every five epochs. Subsequently, the model from the previous round is pitted against each checkpoint model in 300 games, and the best-performing model with a win rate exceeding 55% is selected as the base model for the next round of self-play.

In the details of self-play, the SLM executes 800 parallel MCTS simulations per move. The temperature parameter τ is set to 1 for the first 30 moves of the layout phase to enhance move diversity and is subsequently reduced to $\tau = 0.05$ to ensure that the probability of selecting the most-visited action approaches unity. For the HBM upper network, each move involves 200 parallel MCTS simulations. During the battle phase, the temperature parameter τ is maintained at 1 for the initial 10 moves to promote exploration, after which it is decreased to $\tau = 0.1$ for the remaining moves to prioritize exploitation of the most promising actions. Additionally, during the self-play of the HBM lower network, the initial exploration rate ϵ is set to 0.5 and is gradually decreased over time to balance exploration and exploitation effectively.

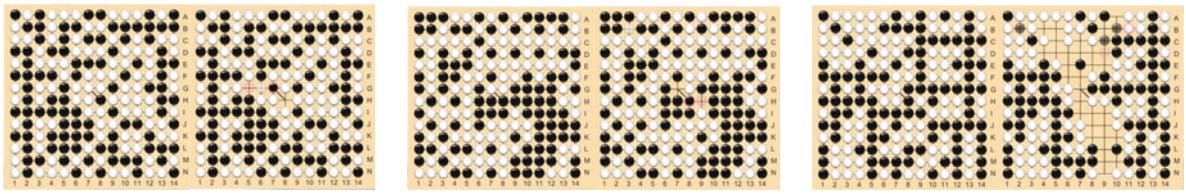
The self-play training was conducted over a 15-day period. The training and self-play configurations of the models are summarized in Table S1. All game data (e.g., time per move) were tested on an NVIDIA RTX 3060 GPU running under the Windows 11 environment.

Table S1 Training and self-play configuration

Configuration	Details
Training GPU	1 NVIDIA RTX 4090D
Training CPU	16 Xeon(R) Platinum 8474C
Training Memory	24 GB
Training Operating System	Ubuntu 22.04.3 LTS
Self-playing GPU	1 NVIDIA RTX 3060
Self-playing CPU	20 Intel(R) Core(TM) i7-12700H
Self-playing Memory	6 GB
Self-playing Operating System	Windows 11
Number of Threads Launched in Parallel MCTS	16

4.3 Rapid Square Formation by SLM in the 2024 National Computer Games Tournament

The primary decision-making objective of the SLM model is to assist the agent in forming squares as quickly as possible during the battle phase, thereby gaining the first-move advantage. Figure S4 presents three sets of images depicting SLM's gameplay during the competition. The first image in each group depicts the game state after SLM has completed its layout, and the second image shows the state when our side forms a square for the first time.



(a) First group of Match Images (b) Second Group of Match Images (c) Third Group of Match Images

Fig. S4 Strategic Layout of SLM During the Match

In Figure S4a, our side plays as White. Upon entering the battle phase, the opponent makes the first

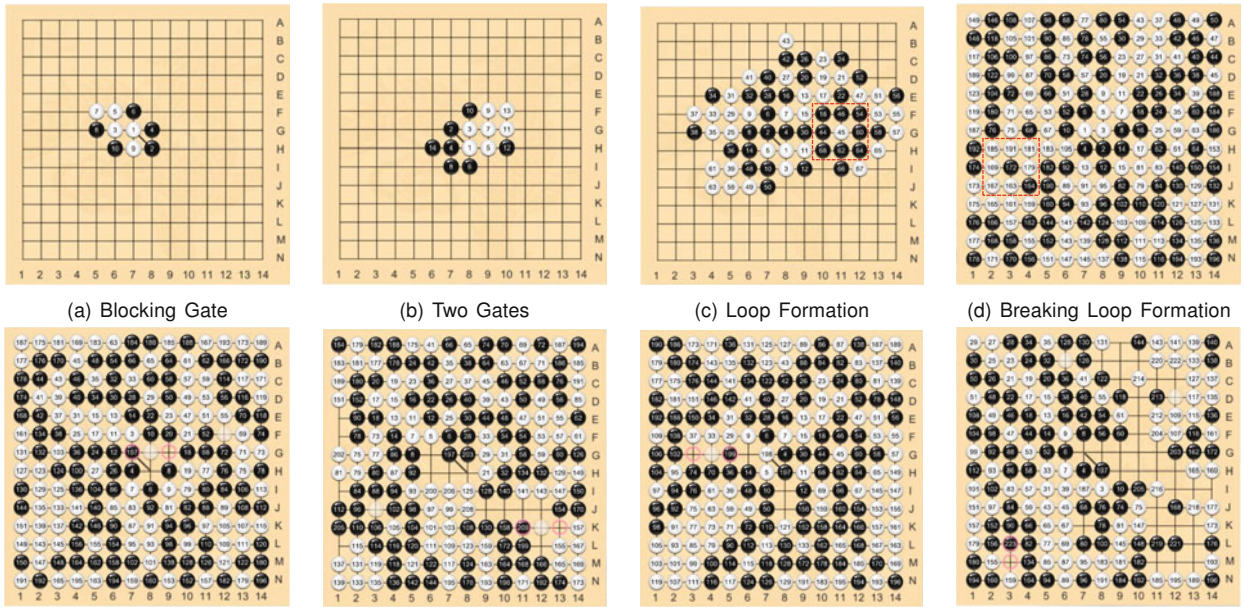
move by jumping from (G,5) to (G,7). Our side's first move can either be moving from (G,8) to (H,8) or from (H,9) to (H,8). During the match, our side opted to move from (G,8) to (H,8), thereby forming a square first.

In Figure S4b, our side plays as Black. Upon entering the battle phase, our side makes the first move by moving from (H,9) to (H,8), forming a square and capturing the opponent's piece at (B,3), thereby establishing a favorable position.

In Figure S4c, our side plays as White. During the battle phase, both sides are evenly matched, and after more than 30 moves, our side jumps from (D,9) to (B,9) and then to (B,11). At (B,11), our side forms a square first, captures the opponent's piece at (B,2), and establishes a *Dalian* formation at (B,2) that the opponent cannot disrupt, thus securing the victory.

4.4 Demonstration of JFA's Advanced Chess Playing Capabilities

The following examples illustrate some of JFA's decisions from the perspective of the black pieces, as shown in Figure S5. Specifically, Figures S5a, S5b, S5c, and S5d depict advanced moves learned during the layout phase, while Figures S5e, S5f, S5g, and S5h demonstrate advanced moves from the battle phase.



(a) Blocking Gate (b) Two Gates (c) Loop Formation (d) Breaking Loop Formation

(e) Square Formation (f) Dalian Formation (g) Two Gates (h) Two Square Formation

Fig. S5 Strategies and Tactics Learned by JFA

From Figures S5a, S5b, S5c, and S5d, it is evident that JFA has grasped the fundamental principles of the Tibetan Jiu chess layout, strategically placing pieces from the center of the board outward. The strategy during the layout phase focuses on forming as many square formations as possible while simultaneously preventing the opponent from doing the same. For instance, in Figure S5a, during the sixth, eighth, and tenth moves, JFA blocks the opponent's gates, thereby preventing the formation of squares. In Figure S5b, JFA places a piece at H6 on the fourteenth move, creating two gates that make it impossible for the opponent to block both simultaneously. Additionally, JFA has mastered complex layout techniques; as shown in the red-boxed area of Figure S5c, JFA surrounds the white piece at G11, forming a loop formation typical in Tibetan Jiu chess. In Figure S5d, JFA disrupts the opponent's attempt to establish a loop formation by defending positions I1 and I5.

From Figures S5e, S5f, S5g, and S5h, it is clear that JFA has learned to leverage piece movements and captures to gain an advantage. For example, in Figure S5e, JFA makes the first move by jumping from G9 to G7, forming a square formation and capturing the white piece at F12. In Figure S5f, JFA

forms a square through jump captures at J3, creating a *Dalian* formation that grants a significant strategic advantage. Figure S5g shows JFA jumping from G3 to G5, establishing two gates that create opportunities for subsequent jump captures and square formations. Lastly, Figure S5h demonstrates JFA forming two square formations through single-step movements, capturing the white pieces at B6 and D12, thereby achieving a combination of offensive and defensive tactics.

These examples collectively demonstrate that JFA, through extensive self-play and reinforcement learning, has developed sophisticated strategies that encompass both the layout and battle phases of Tibetan Jiu chess. Its ability to form and disrupt formations, execute strategic captures, and balance offensive and defensive maneuvers showcases its advanced playing capabilities, approaching the proficiency of skilled human players.

4.5 SLM Robustness Challenges and HBM Decision Latency

1. **Robustness of SLM in Extreme or Rare Game Scenarios**: SLM may exhibit reduced robustness when confronted with extreme or uncommon game situations. For instance, during the layout phase of Tibetan Jiu chess, the typical strategy involves expanding from the center outward. However, if an opponent begins placing pieces from the periphery, SLM tends to follow the opponent's moves rather than occupying advantageous central positions. Future work should focus on increasing data diversity and enhancing the model's generalization capabilities to address this issue.

2. **Decision-Making Time Discrepancy Between HBM and SLM**: HBM experiences significantly longer decision-making times compared to SLM. Future research should explore more efficient algorithms and optimize pruning strategies to further improve the decision-making efficiency of HBM.