Proceedings of the 4th International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/2026.34448

A Hierarchical and Risk-Aware Decision Framework for Riichi Mahjong AI

Haoran Zheng

School of Computing and Data Science, Xiamen University Malaysia, Sepang, Malaysia
CST2309188@xmu.edu.my

Abstract. Riichi Mahjong is a difficult imperfect-information game. To improve the way Mahjong AI makes decisions, this paper proposes a hierarchical and risk-aware decision framework. Instead of choosing an action directly from the current state, the model first selects a high-level strategy and then evaluates specific actions under that strategy. A risk control head is also added so that the model can better handle dangerous situations and make a better trade-off between offense and defense. Experimental results show that this framework improves overall decision quality and gives a better balance between attack and safety. Among the tested models, the version that combines both strategy and risk performs the best overall. The latent strategy analysis also shows that the model learns different internal decision modes with different behavior patterns. This suggests that the proposed method is effective for Riichi Mahjong AI and can make the decision process more structured, stable, and easier to understand.

Keywords: Riichi Mahjong AI, Imperfect-information game, Hierarchical decision framework, Latent strategy modeling, Risk-aware decision-making

1. Introduction

In artificial intelligence, decision-making under uncertainty is an important problem. Games are often used to study it because they have clear rules and clear results. In perfect-information games like Go and Chess, all players can see the full state, so they are easier to model. Many AI systems have already reached very high performance in these games[1-4]. Imperfect-information games are harder because some information is hidden, and players must also respond to other players' decisions.

In recent years, strong progress has been made in imperfect-information games such as Poker. Some systems that combine deep reinforcement learning and regret minimization have performed very well against professional human players [5-7]. Even so, these methods are not easy to apply directly to more complex games. Many real game environments have more players, more actions, and more complicated state changes, so the decision process becomes much harder.

Riichi Mahjong is a very typical example of this kind of difficult environment. It is a four-player imperfect-information game with hidden tiles, randomness, and a very large state space [8]. During the game, a player cannot only think about hand progress. The player also needs to watch opponents, judge danger, and decide when to attack or when to defend.

DOI: 10.54254/2755-2721/2026.34448

Most existing Mahjong AI systems are based on rules, probability methods, or deep reinforcement learning. These methods can reach strong performance, but many of them still use a flat decision style. In other words, they try to choose an action directly from the current state in one step. This works to some extent, but it is not very close to how human players usually think [9]. It also makes the model harder to explain, because the system gives an action but does not clearly show the reason behind it.

Human players usually do not think in such a direct way. In many cases, they first decide on a general direction, such as pushing forward, defending, or using a more balanced style. After that, they choose a specific action that matches this direction [10]. So the decision process is often hierarchical. There is first a high-level idea, and then a low-level move. This kind of structure may be more suitable for Mahjong AI, because it is closer to actual human play and may also make decisions clearer and more flexible.

Based on this idea, this paper proposes a hierarchical and risk-aware decision framework for Riichi Mahjong AI. The method splits decision-making into two parts: high-level strategy selection and low-level action execution. At the same time, it adds risk awareness, so the model can better handle the choice between offense and defense. The main purpose is to make Mahjong AI more consistent, easier to understand, and better able to deal with complex game situations.

2. Problem formulation

2.1. Riichi Mahjong as an AI benchmark

Mahjong is a typical imperfect-information game. Players cannot see all the information, luck affects the game, and four players interact all the time. Among Mahjong variants, Riichi Mahjong is the one most often used in AI research. One important reason is the availability of large online game records, such as Tenhou[11, 12], which provide useful data from strong human players [13].

Because of these data, Riichi Mahjong has become a practical benchmark for AI research. It can be used for both supervised learning and self-play reinforcement learning. Several strong Mahjong AI systems, such as Suphx [14], Naga [15] and LuckyJ [16], also show that deep reinforcement learning and neural networks can work well in this field.

2.2. Game structure of Riichi Mahjong

Riichi Mahjong uses 136 tiles, including three suits and honor tiles. Each player starts with 13 tiles. The main goal is to complete a winning hand, usually four melds and one pair. A player can win by self-draw (Tsumo) or by another player's discard (Ron). Actions such as Chow, Pong, and Kong can also be used during the game, and they directly affect later decisions.

The Riichi declaration is another important part of the game. When a player reaches a ready hand state, the player may declare Riichi. This gives possible scoring benefits, but it also makes the hand direction more fixed.

2.3. Player interaction and strategic decomposition in mahjong

Mahjong includes many small tactical choices [17]. For attack, players may use Chow or Pong to speed up the hand or improve shape. For defense, they may choose safer tiles and judge danger from visible information.

DOI: 10.54254/2755-2721/2026.34448

These choices keep changing during the round. A good move early may not be good later. So players must keep adjusting based on hand quality, round stage, and opponent behavior. In real games, players often decide a general direction first, such as attack, defense, or a balanced style, and then choose a specific move.

Because of this, Mahjong decisions naturally have two levels: strategy first, action second. These strategic ideas are not directly written in the state itself. They come from reading the whole table and judging the situation. This is also the reason for using a hierarchical and risk-aware framework in this work, where high-level strategy is separated from low-level action selection.

3. Related work

3.1. Rule-based and heuristic approaches

Early Mahjong AI was mainly built with rules. The idea was to summarize human playing experience and turn it into fixed decision rules. These rules usually focused on tile efficiency, hand value, and basic defense.

This kind of method can work, but it also has clear limits. Mahjong situations change quickly, and fixed rules are often too rigid. They also do not adapt well to different player styles, so their performance is limited in stronger games.

3.2. Supervised learning from expert data

Later, researchers used supervised learning instead of relying only on hand-written rules. The model was learned from real match records played by strong human players. However, supervised learning also has weaknesses. Its performance depends heavily on the quality and diversity of the training data. It mainly learns to imitate past decisions, so it does not always capture the best long-term strategy. Because of this, performance may become unstable in unfamiliar situations.

3.3. Deep reinforcement learning in Mahjong AI

Deep reinforcement learning greatly improved Mahjong AI. A typical example is Suphx, which reached a very high level through large-scale self-play. This method allows the model to learn from repeated games and improve decisions based on long-term results, not only one-step outcomes.

Still, Mahjong remains difficult for DRL. Much important information is hidden; one action may affect the game many turns later, and the game involves four interacting players. These factors make training more complex. In addition, DRL often needs large amounts of data, long training time, and strong computing resources, which makes it expensive in practice.

3.4. Advanced neural architectures for feature representation

Many studies also tried to improve Mahjong AI by improving state representation. The basic idea is that if the model understands the game state better, it may also make better decisions. Some work used convolution-based models such as Res2Net to learn spatial features in Mahjong states [18].

Other work focused on sequential information. LSTM was used to keep track of previous steps and provide more context[19]. Later, Transformer-based models such as Tjong were introduced to capture more complex long-range relations [20].

These models improve how the state is represented, but in many cases, they still keep a flat decision pattern. The model reads the current state and directly outputs an action, so the overall

DOI: 10.54254/2755-2721/2026.34448

decision logic is not fundamentally changed.

3.5. Game-theoretic and hybrid approaches

Besides pure learning methods, some studies also use ideas from game theory to make Mahjong AI think better. For example, Counterfactual Regret Minimization (CFR) has been used in Mahjong-like games, helping the AI balance attack and defense better. Systems like LuckyJ combine neural networks with CFR-based search and get good results with a better strategic balance [16].

Some actor-critic methods, such as Actor-Critic Hedge (ACH), also try to improve reinforcement learning by making training more stable. However, these methods still need a lot of computing resources, and they may still have trouble using long game flow and context information well in Mahjong.

3.6. Limitations of existing approaches and motivation for this work

Although Mahjong AI has improved a lot, several problems remain. Many methods still need large amounts of data and computation. Also, even when state representation becomes better, the decision process itself often stays flat.

More importantly, most systems still go directly from state to action. They do not first form a strategy and then choose a move. This makes the decision process harder to explain and sometimes less flexible in complex situations.

To address this, this work proposes a hierarchical and risk-aware framework for Riichi Mahjong AI. It first selects a strategy, then chooses an action, and also considers risk. The goal is to make the model closer to human Mahjong thinking and more reliable in difficult situations.

4. Methodology

This study builds a hierarchical and risk-aware decision framework for Riichi Mahjong. The main idea is simple. Good Mahjong play should not be treated as a direct jump from state to action. There should be a middle step for strategy, and there should also be a way to control risk when the board becomes dangerous. So the model is built with three main parts: a shared state representation network, a discrete strategy module, and a risk control head. Fig. 1 shows the overall architecture of the proposed hierarchical and risk-aware decision framework.

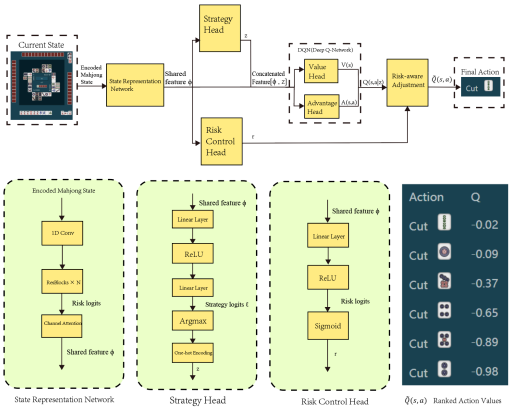

Figure 1. Overall architecture of the proposed hierarchical and risk-aware Riichi Mahjong decision framework (picture credit: original)

DOI: 10.54254/2755-2721/2026.34448

Given an encoded Mahjong state s, the model first extracts a shared feature ϕ. From this feature, the strategy branch predicts a discrete strategy mode z, and the risk branch outputs a scalar risk score r. These signals are then used in action-value estimation. The whole process can be written as

$$\phi = F(s), z = G(\phi), r = R(\phi), Q(a \mid s, z) = H([\phi; z], a) \tag{1}$$

Here, F(·) is the shared encoder, G(·) is the strategy head, R(·) is the risk head, and H(·) is the action-value network. In this design, strategy is not just an extra label for explanation. It directly affects later action evaluation.

The shared representation network is made of one-dimensional convolution, residual blocks, and channel attention. The convolution layer captures local patterns in the encoded Mahjong state. The residual blocks help the model learn deeper features without making training too unstable. Channel attention then reweights feature channels so that more useful information gets stronger emphasis. The final output is the latent feature ϕ, which is shared by the later modules.

4.1. Action-value estimation

To score legal actions, the model uses a dueling value structure. This means the network separates two things: the overall value of the current state and the relative advantage of each action. This fits Mahjong well, because many legal actions come from the same board situation but still differ in quality.

Given the strategy-conditioned feature [ϕ;z], the model predicts a scalar state value V and an action-advantage vector A. The final action value is computed as

$$Q\left(s, a \mid z\right) = V\left(\left[\phi; z\right]\right) + A\left(\left[\phi; z\right], a\right) - \frac{1}{|A|} \sum_{a' \in \mathcal{A}} A([\phi; z], a') \tag{2}$$

Here, A is the legal action set. This helps the model keep a stable estimate of how good the whole state is, while also comparing the available actions more carefully. Since z is concatenated with ϕ, the action-value network is guided by a high-level strategy mode instead of relying only on raw state features.

4.2. Risk control head

The risk control head is an important part of the model. In Mahjong, bad decisions are often not caused by missing good attacking chances. More often, the problem is failing to stop dangerous actions when the threat is imminent. To deal with this, the model adds a separate risk branch that predicts a danger signal from the shared feature.

The risk head takes ϕ as input and outputs a scalar risk score r∈[0,1]. This score is used to lower the scores of aggressive actions during final action selection. Let g(a) be an action-dependent aggressiveness coefficient. The risk-aware adjusted score is

$$\widetilde{Q}\left(s, a\right) = Q\left(s, a \mid z\right) - \lambda_r \, r \, g\left(a\right) \tag{3}$$

DOI: 10.54254/2755-2721/2026.34448

where λ_r controls how strong the risk penalty is. So the model does not lower every action in the same way. It mainly suppresses actions that are too aggressive for the current situation.

This makes risk a clear part of the model structure. Instead of hoping the action-value network will learn all defensive behavior by itself, the model uses a separate branch to estimate danger and adjust action scores directly.

4.3. Discrete strategy module

The strategy module is the high-level layer of the model. Its job is to divide Mahjong states into a small number of latent behavior modes before detailed action scoring starts. In the final model, the strategy variable is a fixed discrete variable $z \in \{0,1,2\}$. These three modes are learned automatically during training.

Given the shared feature ϕ, the strategy head first outputs a three-dimensional logit vector.

$$l = G_{\psi}(\phi), l \in \mathbb{R}^3 \tag{4}$$

where G_ψ (·) is the strategy prediction branch. During inference, the model selects the dominant strategic mode through hard assignment,

$$k^* = arg\max_{k \in \{0,1,2\}} l_k, z = onehot\left(k^*\right) \tag{5}$$

This means each state is first mapped to one of three latent strategy categories, and that selected category is then used in later action evaluation.

To keep the model trainable while still using discrete strategy output, the forward pass uses hard assignment, while training uses a differentiable approximation. In implementation, this is done with a straight-through discrete relaxation. So gradients can still pass through the strategy branch even though the output behaves like a one-hot code. This lets the model learn different strategy modes without using manually labeled strategy classes.

After the discrete strategy code is obtained, it is concatenated with the shared state feature and sent into the action-value network:

$$h = [\phi; z], Q(s, a \mid z) = H_{\eta}(h, a) \tag{6}$$

This is one of the key parts of the whole design. It shows that strategy is not just used for explanation after the action is chosen. It is part of the value computation itself. So the same Mahjong state may lead to different action preferences under different strategy modes.

This is also why the framework is hierarchical. The shared encoder first summarizes the board state. Then the strategy module decides the high-level mode. After that, the action-value network scores actions under that strategy context. This gives a more structured way to model different styles of play.

DOI: 10.54254/2755-2721/2026.34448

4.4. Training objective

All parts of the model are trained together end-to-end. The main goal is still action-value learning, which makes the predicted value of the chosen action match the return target from the game outcome. At the same time, extra terms are added for conservative regularization, auxiliary supervision, latent strategy regularization, and risk prediction.

The final training objective is

$$L = \mathcal{L}_Q + \lambda_c \mathcal{L}_{cons} + \lambda_a \mathcal{L}_{aux} + \mathcal{L}_{str} + \lambda_r \mathcal{L}_{risk} \tag{7}$$

Here, $\mathcal{L}_Q$ is the action-value regression loss, $\mathcal{L}_{\text{cons}}$ is the conservative regularization term, $\mathcal{L}_{\text{aux}}$ is the auxiliary prediction loss, $\mathcal{L}_{\text{str}}$ is the latent strategy regularization term, and $\mathcal{L}_{\text{risk}}$ is the risk prediction loss. The latent strategy regularization is used to prevent the collapse of the discrete strategy variable and to encourage non-degenerate usage of the available strategic modes.

With this joint objective, the model learns not only how to score actions, but also how to form stable strategy patterns and how to control risk. In the end, the whole system works as a hierarchical Mahjong decision model where representation learning, strategy inference, risk control, and action evaluation are trained together.

5. Experiments and analysis

5.1. Experimental setup

All experiments were run on a machine with an NVIDIA GeForce RTX 4070 SUPER GPU. Each model was tested over 2000 games. The test setting was kept the same for all models so the results could be compared fairly.

In this paper, the baseline model refers to the original Mahjong decision model before the proposed strategy and risk components are added. It follows a direct state-to-action value process, where the encoded game state is mapped to action scores through shared feature extraction and action-value estimation.

For the ablation study, each non-baseline model played against three identical baseline agents. The baseline itself was measured by letting four baseline agents play against each other and then averaging the results. This setup makes the comparison simple and stable. It also helps show whether a change in the model really improves play, instead of only working under a special opponent mix.

The table reports two kinds of information. One part is about overall performance. This includes average rank and average point gain. These show whether a model does better across full games. The other part is about playing style. Win rate, average winning point, and point efficiency describe offense. Deal-in rate, average deal-in point, and deal-in loss describe defense. Riichi rate, fuuro rate, and tsumo rate help show how the model tends to play. Together, these metrics give a clearer picture than rank alone.

5.2. Ablation study

The overall comparison results of different model variants are shown in Table 1.

DOI: 10.54254/2755-2721/2026.34448

Table 1. Ablation results of different model variants in 2000-game evaluation

	avg_ rank	avg_ pt	agari_ rate	avg_agari_ point	point_ efficiency	houjuu_ rate	avg_houjuu_ point	houjuu_ loss	riichi_ rate	fuuro_ rate	tsumo_ rate
Baseline	2.50	0.00	21.44	7155.82	1534.33	13.77	5249.33	722.87	26.62	15.53	35.78
Add_Risk	2.42	3.60	21.59	7223.77	1559.44	11.23	5370.69	603.10	28.51	20.72	34.75
Add_Strategy	2.47	6.30	21.23	6409.62	1360.49	10.26	5845.63	600.00	14.50	13.95	37.79
Add_Risk_ Strategy	2.39	6.08	20.24	7536.61	1525.31	11.19	5564.00	622.53	23.47	18.80	43.49

Add_Risk mostly helps on the defensive side. The biggest change is that the model deals less often, and when it does, the total loss is smaller. That is exactly what the risk head was supposed to do. At the same time, the offense does not fall apart. The model still wins at about the same level, and in some places looks slightly better than the baseline. So this is not just a “more passive” model. It is still willing to attack, but it seems to make fewer bad attacks.

Add_Strategy looks different. Its results are better than the baseline overall, but the path is not the same as Add_Risk. The strategy model does not mainly win by pushing harder. Instead, it seems to play in a more filtered way. Some risky actions disappear, some over-commitment goes down, and the whole style becomes more selective. In Mahjong terms, this looks like a model that starts to separate different board situations instead of treating every hand in the same way.

The best overall result in the table comes from Add_Risk_Strategy. This is the most important part of the ablation study. It suggests that the two added parts are helping in different places. The risk head helps the model stay safer. The strategy part changes how the model thinks about the situation before choosing an action. When both are used together, the result is more balanced. The model not only avoids mistakes better. It also seems better at choosing when to push and when not to.

So the ablation result is fairly clear. Risk control helps safety. Strategy changes decision style. Putting them together gives the strongest final behavior.

5.3. Latent strategy analysis

The behavioral statistics of the learned latent strategy states are shown in Table 2. The table looks at the latent strategy variable z. This part matters because the strategy module is one of the main ideas in the model. If the three values of z all behaved the same way, then the strategy layer would not mean much. But that is not what the table shows.

Table 2. Behavioral statistics of the learned latent strategy states

z	samples	win_rate	avg_win_pt	houjuu_rate	avg_houjuu_pt	pass_rate	tsumo_given_win
0	26598	7.53	9820.93	15.76	5582.22	26.59	51.16
1	20372	10.43	8875.00	9.66	5719.05	15.96	35.29
2	17873	40.65	5411.75	6.71	6567.31	10.82	36.51

First, all three values of z are used many times. That already matters. It means the model is not collapsing everything into one class. The strategy layer is actually active.

DOI: 10.54254/2755-2721/2026.34448

Second, the three groups look clearly different. One group has fewer wins, but those wins are worth more, and it also shows higher risk. That looks like a more dangerous, high-reward style. Another group sits more in the middle. It is not the safest and not the wildest either. The last group wins much more often, has lower risk, but those wins are not as large. That looks more like a stable and efficient style.

This matters because the model was not given hard human labels like “attack” or “defense” during training. Those patterns appeared from the learned z values themselves. So the table is not just showing three random buckets. It is showing three different ways of playing.

The difference between the three z groups is not small. They do not just shift a little around the same behavior. They show different trade-offs between risk, hand value, and winning frequency. One mode looks more reward-seeking. One looks more balanced. One looks safer and more efficient. That is exactly the kind of split expected from a strategy module.

This is also why the result is meaningful beyond the raw score table. The final model is not only getting better numbers. It is also organizing its decisions in a more structured way. The strategy variable is doing something real inside the model. It is not just an extra output added for show.

Taken together, the two tables support the same conclusion. The added modules not only improve performance. They also change how the model plays. Risk control makes the model safer. The strategy layer separates different kinds of game situations into different modes. The full model benefits from both.

6. Conclusion

This paper proposed a hierarchical and risk-aware decision framework for Riichi Mahjong AI. The main idea was to move away from a flat state-to-action style and add a clearer middle step for strategy. At the same time, a risk control branch was added so the model could respond better when the table became dangerous. In this way, the whole decision process became more structured and closer to actual Mahjong thinking.

The experimental results showed that these two parts played different roles. The risk module mainly helped reduce unsafe decisions and improve defensive stability. The strategy module helped separate different game situations into different decision modes, instead of treating all states in the same way. When the two parts were combined, the final model achieved the best overall behavior. It not only improves performance, but also shows a clearer balance between offense and defense.

The latent strategy analysis also supported this idea. The learned strategy variable did not collapse into one meaningless category. Different strategy states showed different patterns in win rate, hand value, and risk. This suggests that the model really learned an internal strategy layer, not just a better action scorer.

Overall, the result shows that adding strategy and risk into the decision process is useful for Mahjong AI. It helps the model play in a way that is more stable, more flexible, and easier to understand. In the future, this framework can be extended with richer strategy types, stronger risk modeling, and larger-scale training and evaluation.

References

[1] Silver, D., 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

[2] Silver, D., et al. (2017). Mastering the game of go without human knowledge. Nature, 550(7676), 354–359. https: //doi.org/10.1038/nature24270

DOI: 10.54254/2755-2721/2026.34448

[3] Silver, D., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv. https: //doi.org/10.48550/arXiv.1712.01815
[4] Schrittwieser, J., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. Nature, 588(7839), 604–609. https: //doi.org/10.1038/s41586-020-03051-4
[5] Moravčík, M., et al. (2017). DeepStack: expert-level artificial intelligence in heads-up no-limit poker. Science, 356(6337), 508–513. https: //doi.org/10.1126/science.aam6960
[6] Brown, N., & Sandholm, T. (2017). Libratus: The superhuman AI for no-limit poker. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (pp. 5226–5228). International Joint Conferences on Artificial Intelligence Organization. https: //doi.org/10.24963/ijcai.2017/772
[7] Brown, N., & Sandholm, T. (2019). Superhuman AI for multiplayer poker. Science, 365(6456), 885–890. https: //doi.org/10.1126/science.aay2400
[8] Zhang, R., et al. (2025). A survey on self-play methods in reinforcement learning. arXiv. https: //doi.org/10.48550/arXiv.2408.01072
[9] Zheng, Y., & Li, S. (2020). A review of mahjong AI research. In Proceedings of the 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence (pp. 345–349). ACM. https: //doi.org/10.1145/3438872.3439104
[10] Sato, H., Shirakawa, T., Hagihara, A., & Maeda, K. (2017). An analysis of play style of advanced mahjong players toward the implementation of strong AI player. International Journal of Parallel, Emergent and Distributed Systems, 32(2), 195–205. https: //doi.org/10.1080/17445760.2015.1049267
[11] Yamaguchi, Y. (2024). An overview of online riichi mahjong game industry in japanese internet society. In 2024 IEEE Gaming, Entertainment, and Media Conference (GEM) (pp. 1–3). https: //doi.org/10.1109/GEM61861.2024.10585787
[12] Tenhou | A Top-Level Online Mahjong Game Site. (n.d.). Retrieved March 20, 2026, from https: //tenhou.net/
[13] Louie. (2026, March 13). NikkeTryHard/tenhou-to-mjai. GitHub. Retrieved March 19, 2026, from https: //github.com/NikkeTryHard/tenhou-to-mjai
[14] Li, J., et al. (2020). Suphx: mastering mahjong with deep reinforcement learning. arXiv. https: //doi.org/10.48550/arXiv.2003.13590
[15] Dwango Media Village. (n.d.). Retrieved March 19, 2026, from https: //dmv.nico/
[16] Oriol, M. (n.d.). LuckyJ: an asynchronous evolution platform for component-based applications.
[17] What Is Mahjong? A Free Online Mahjong Strategy Guide. (n.d.). Retrieved March 20, 2026, from http: //beginners.biz/kihon/kihon01.html
[18] Li, X., Dai, J., Su, C., Fan, X., & Gu, J. (2026). Efficient mahjong model using an improved distributed proximal policy optimization algorithm and residual residual network-integrated long short-term memory network. Engineering Applications of Artificial Intelligence, 166, 113475. https: //doi.org/10.1016/j.engappai.2025.113475
[19] Li, X., Wang, Z., Liu, B., & Dai, J. (2024). LsAc*-MJ: A low-resource consumption reinforcement learning model for mahjong game. International Journal of Intelligent Systems, 2024(1), 4558614. https: //doi.org/10.1155/2024/4558614
[20] Li, X., Liu, B., Wei, Z., Wang, Z., & Wu, L. (2024). Tjong: A transformer-based mahjong AI via hierarchical decision-making and fan backward. CAAI Transactions on Intelligence Technology, 9(4), 982–995. https: //doi.org/10.1049/cit2.12298