Building a 3-Player Mahjong AI using Deep Reinforcement Learning

Xiangyu Zhao
Department of Computer Science and Technology
University of Cambridge
Cambridge, United Kingdom
xz398@cam.ac.uk

Sean B. Holden
Department of Computer Science and Technology
University of Cambridge
Cambridge, United Kingdom
sbh11@cl.cam.ac.uk

***Abstract*—Mahjong is a popular multi-player imperfect-information game developed in China in the late 19th-century, with some very challenging features for AI research. Sanma, being a 3-player variant of the Japanese Riichi Mahjong, possesses unique characteristics including fewer tiles and, consequently, a more aggressive playing style. It is thus challenging and of great research interest in its own right, but has not yet been explored. In this paper, we present Meowjong, an AI for Sanma using deep reinforcement learning. We define an informative and compact 2-dimensional data structure for encoding the observable information in a Sanma game. We pre-train 5 convolutional neural networks (CNNs) for Sanma's 5 actions—discard, Pon, Kan, Kita and Riichi, and enhance the major action's model, namely the discard model, via self-play reinforcement learning using the Monte Carlo policy gradient method. Meowjong's models achieve test accuracies comparable with AIs for 4-player Mahjong through supervised learning, and gain a significant further enhancement from reinforcement learning. Being the first ever AI in Sanma, we claim that Meowjong stands as a state-of-the-art in this game.**

***Index Terms*—Mahjong, deep learning, reinforcement learning, convolutional neural networks, policy gradient methods**

I. Introduction

Mahjong is a popular tile-based multi-round multi-player imperfect-information game that was developed in China in the late 19th century, and nowadays has hundreds of millions of players worldwide. It is a game of skill, strategy and calculation, and involves a degree of chance. Mahjong is very challenging for AI because:

- Mahjong is played by more than 2 players and has significant hidden information;
- Mahjong has complex playing and scoring rules;
- Mahjong has a huge number of possible winning hands in various patterns, allowing flexible in-game strategy adaptation.

While there are many variants of Mahjong, we focus on Sanma, a 3-player variant of the Japanese Riichi Mahjong. Sanma differs fundamentally from 4-player Japanese Riichi Mahjong because most of the tiles in a suit are removed, and one of the actions (Chii) is replaced with another action (Kita). Consequently, both the number of possible states and the amount of hidden information are reduced, and hands tend to develop much faster. As a result, players tend to play more aggressively, and valuable winning hands become more frequent. Since it is commonly understood by expert human players that the 4-player Japanese Riichi Mahjong should be played defensively, an AI trained to learn optimal strategy for the 4-player Japanese Riichi Mahjong is not guaranteed to acquire the optimal strategy for Sanma. Therefore, even though Sanma is seemingly simpler than 4-player Mahjong, it still preserves all the challenging characteristics of Mahjong, and the difference in strategy also makes it a worthwhile target for research.

In this paper, we describe Meowjong, a Sanma AI based on deep reinforcement learning (DRL). First, we propose a 2-dimensional data structure for encoding the observable information in a round of a Sanma game. We then pre-train 5 deep convolutional neural networks (CNNs), each corresponding to one action in Sanma (discard, Pon, Kan, Kita, and Riichi), and improve the discard model using the Monte Carlo policy gradient method for self-play reinforcement learning (RL).

II. Related Works

As far as we are aware, there has not been any published attempt to develop an AI player for Sanma. Various machine learning approaches have been tried on 4-player Mahjong, including Bakuuchi by Mizukami et al. [1] in 2015, which was based on Monte Carlo simulation and opponent modelling. Bakuuchi has achieved a rating of 1718 on Tenhou [2] (a popular global online Riichi Mahjong platform with more than 350,000 active users); this was significantly higher than that of an average human player. In 2019, Kurita et al. [3] built a Mahjong AI based on multiple Markov decision processes as abstractions of Mahjong, and also reached the top level of Mahjong AI, as demonstrated by playing directly against Bakuuchi. In 2018, Gao et al. [4] built a Mahjong AI using CNNs and achieved a test accuracy of 68.8% for the discard action, which is 6.7% higher than the previous best test accuracy of 62.1% reported by Bakuuchi. They also evaluated the AI's strength on Tenhou, reaching a rating of 1822 after 76 matches. In 2019, Li et al. [5] built a Mahjong AI called Suphx, based on DRL. Suphx parameterized its policy using CNNs, and improved its discard model through distributed RL. It also introduced global reward prediction based on gated recurrent units (GRUs) to predict the final reward of an entire game based on the information of the current and previous

rounds. Furthermore, it introduced oracle guiding to speed up the agent's improvement in RL. During online playing, Suphx employed run-time parametric Monte Carlo policy adaptation to exploit the new observations on the current round in order to perform even better. Suphx eventually reached a stable rank of 8.74 dan on Tenhou, which is about 2 dan higher than Bakuuchi, and is higher than 99.99% of all the officially ranked human players on Tenhou, though at a cost of needing extremely heavy computational resources for training. While the performance of these systems is impressive, we emphasize that, as we are unaware of comparable work addressing Sanma, no such comparisons with human players are available in this context.

III. Basic Rules and Terminology of Sanma

Sanma has 108 *tiles* of 27 different kinds, with four identical tiles in each kind. TABLE I shows the appearances of the 27 kinds of tiles. These 27 kinds are:

- *Manzu*: 1m (Man) and 9m, with traditional Chinese character patterns;
- *Pinzu*: 1–9p (Pin), with dot patterns;
- *Souzu*: 1–9s (Sou), with bamboo patterns (1s is often decorated with a bird);
- *Honours*: this includes four *winds*: East, South, West, North, and three *dragons*: Haku (white), Hatsu (green), Chun (red).

At the start of each round, the tiles are shuffled and arranged into *walls* of face-down two-tier-high tiles. Each player receives 13 tiles from the wall as their starting *hand*. The last 14 tiles form the *dead wall*, which will not be used throughout the current round except in some specific cases, and the rest of the walls with available tiles form the *live walls*. The seats of the players are denoted East, South and West, also referred to as the players' *seat winds*. The East player in each round becomes the dealer. In each round, the players take turns to draw and discard tiles until one of them completes a winning hand. The round ends with a draw if all tiles are drawn and discarded. Other than the normal discard action, a player can perform the following special actions in legal situations, which may interrupt the regular playing order:

- *Pon*: a player can claim an immediately discarded tile from any other player to form a *triplet* of the same kind (also called a *Koutsu*). This action may cause a player's turn to be skipped. The player needs to discard a tile after calling Pon.
- *Kan*: a player can call Kan to form a *quad* of the same kind (also called a *Kantsu*). The player draws another tile and continues their turn after a Kan, and a *Dora indicator* is also revealed from the dead wall. A Dora indicator indicates corresponding *Doras*; these are bonus tiles that add value to a winning hand.
- *Kita*: a player can put a North tile to the side of their hand, and it would be counted as *Dora*, called the *Nukidora*. The player is awarded an extra draw and can continue their turn.

TABLE I
Mahjong tiles[a]

	Numbers								
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Manzu									
Pinzu									
Souzu									
Honours	Winds				Dragons				
	East	South	West	North	Haku	Hatsu	Chun		
									

[a] Graphical resources adapted from FluffyStuff's GitHub repository (https://github.com/FluffyStuff/riichi-mahjong-tiles).

- *Riichi*: a player can pay a 1,000 point deposit and declare a ready hand, which means they only need one more tile to win. After declaring Riichi, the player's hand cannot be changed, and the player must discard any tile they draw until the round ends. The Riichi player bares certain risks, but has an enhanced chance to win and can gain a larger score if they do.

Knowing when to appropriately make those actions is one of the central strategies in Riichi Mahjong.

There are two types of winning: winning from another player's discard (*Ron*), or winning from self-draw (*Tsumo*). In the case of Ron, the player who feeds the winning tile pays the winning player by the amount of the winning player's hand score. In the case of Tsumo, the other two players split the winning player's hand score and pay the winning player together. The dealer wins 50% more, but when a non-dealer player wins by Tsumo, the dealer also pays twice compared to the other paying player.

IV. Features and Data Structure Design

Unlike other board games such as Chess and Go, the layout of a Mahjong board is not standardized, and the observable information must be carefully encoded in order to be digested by the CNNs. Since there are 34 different tiles in Mahjong, we used a 34×366 array to represent a state. Although 2m–8m are not included in Sanma, which leaves only 27 of the 34 tiles to be used, we still included all the tiles and left the excluded tiles blank, to add transferability of Meowjong to 4-player Mahjong. The mapping between the tiles and their corresponding row indices in the array encoding is shown in TABLE II.

To simulate the real in-game environment and maximize Meowjong's performance, all observable information should be taken into account. Therefore, we use 366 columns in total to represent 22 features, as listed in TABLE III. We did not include the number of remaining tiles, because it can be calculated from the number of discarded tiles. For triplets, quads and Riichi status, we include not only the triplet/quad tiles and the Riichi status, but also the turn numbers of the

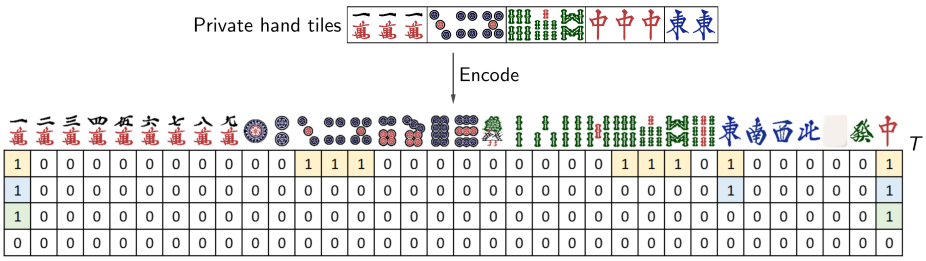

Fig. 1. Encoding of an example private hand using 4 channels (transposed to save space)

TABLE II
TILES AND THEIR CORRESPONDING ROW INDEX REPRESENTATIONS

Tiles	Corresponding indices
Manzu, i.e., 1m–9m	0–8
Pinzu, i.e., 1p–9p	9–17
Souzu, i.e., 1s–9s	18–26
Winds, i.e., East, South, West, and North	27–30
Honours, i.e., Haku, Hatsu, and Chun	31–33

TABLE III
INPUT FEATURES FOR THE MODELS

Feature	Number of channels
Target tile	1
Self private tiles	4
Red Doras[1] in hand	1
Self open triplets/quads	$(4+5)\times 4 = 36$
Self Kitas	4
Self discards	30
Dora indicators	5
Other players' Riichi status	$(1+5)\times 3 = 18$
Scores	$11\times 4 = 44$
Round (Kyoku) number	4
Repeats count (i.e., Honba number)	4
Deposit count	4
Self wind	1
Other players' open tiles[2] and discards	$(36+4+30)\times 3 = 210$
Total	366

Pon/Kan/Riichi calls. Besides, although the spaces for the fourth player are not needed for Sanma, they are still created for the sake of transferability of Meowjong to the 4-player Mahjong.

The included features can be divided into two categories:

- *Tile features* that involve sets or sequences of tiles, such as private tiles, triplets/quads, and so on. Tile features can be encoded by setting the corresponding row indices to 1 and leaving the rest to 0, as, for example, shown in Fig. 1.
- *Numerical features*, for example, player scores. Numerical features can be binary-encoded into multiple columns, each being either all 0's or all 1's.

[2]In Mahjong, one of each of the fives in Manzu, Pinzu and Souzu are marked red and count as Doras, called the *red doras* (also known as the *Akadoras*). They are still the same tiles as the ordinary fives, except that they are worth extra points in the score calculation.

[2]Triplets, quads, and Kitas.

V. NEURAL NETWORK STRUCTURE DESIGN

We parameterize Meojong's policy on an action basis: we use a separate model, called an *action model*, to parameterize Meojong's policy for each action (discard, Pon, Kan, Kita, and Riichi). For the action models of Meowjong, we adopt a CNN structure with 4 convolutional layers followed by a fully-connected layer. Each of the first 3 convolutional layers has 64 filters, and the last convolutional layer has 32 filters. We have experimented with various choices for the model structures, including the number of hidden layers and the number of filters in each layer, and found that having more convolutional layers or more filters per layer could result in an oversized model under our memory constraint. Having fewer convolutional layers would result in a huge number of parameters in the flatten operation before the fully-connected layer, also leading to an oversized model. All filters in the 4 layers share the same size, which is a hyperparameter to be tuned individually for each action model. The fully-connected layer contains 256 perceptrons. A batch normalization (BN) layer and a dropout layer with dropout rate 0.5 are added after each convolutional and fully-connected layer, in order to prevent over-fitting. All action models share a similar structure, differing in the filter sizes and the output dimensions (34 for the discard network, and 2 for the rest). ReLU is used for the activation function of all the convolutional and fully-connected layers, and softmax is used for the activation function of the output layers. The CNN structure for the action models is shown in Fig. 2. The input and output dimensions of the models are shown in TABLE IV.

Since the decision-making problems in Sanma can be converted to multi-class classification problems, we define the loss for our CNNs to be the categorical cross-entropy:

$$L(\mathbf{w}) = -\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \cdot \log \hat{y}_k^{(i)} \tag{1}$$

where m is the number of examples, and K is the number of classes. Although pooling is recognized as an efficient down-sampling tool for CNNs in computer vision tasks due to the shift-invariance property of image recognition, in Meowjong's case, the data structure is not an image, but a compact encoding of discrete feature data, and we would expect the use of pooling here to lose too much information, leading to a lower accuracy. Therefore, pooling is not used in Meowjong's CNN structure. No padding is used in any convolutional layer.

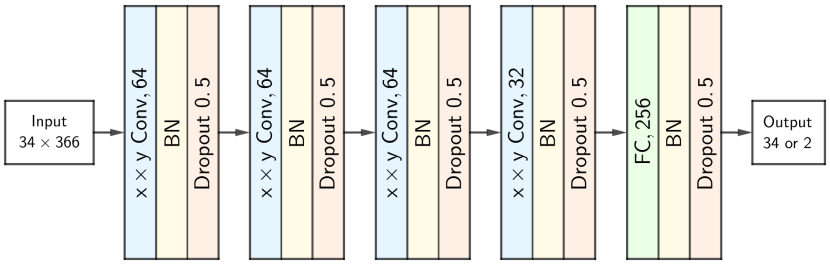

Fig. 2. CNN structure of the action models ($x \times y$ denotes the filter size)

TABLE IV
INPUT AND OUTPUT DIMENSIONS OF THE ACTION MODELS

	Discard	**Pon**	**Kan**	**Kita**	**Riichi**
Input	34×366				
Output	34	2			

TABLE V
FILTER SIZES (x, y) OF THE ACTION MODELS

	Discard	**Pon**	**Kan**	**Kita**	**Riichi**
Filter size (x, y)	$(4, 5)$	$(5, 4)$	$(2, 3)$	$(3, 2)$	$(3, 4)$

The discard problem can be interpreted as a 34-class classification problem, since there are 34 kinds of tile in total, or a 14-class classification problem, since each player can have at most 14 tiles in their private hand in any given state. As recommended by both Li et al. [5] and Gao et al. [4], we adopt the 34-class classification interpretation, and hence use a 34-dimensional output. The only potential risk of the 34-class method is of illegal discards, where the discard model outputs a tile that does not exist in the player's private hand. However, in practice all discard choices made by our discard model are legal; this suggests that our CNN has strong learning ability. For all the rest of the actions, a player can either declare or skip that action in appropriate situations, and all those actions can be interpreted as binary classification problems. Hence, we set up a 2-dimensional output for each of their action models.

Hyperparameter tuning for the pre-training of the CNNs is focused on the filter size for each action model. As there is no guarantee that the same filter size can work for every action, the filter sizes are tuned individually for each action model. Grid search is used for hyperparameter tuning: for each action, all candidate filter sizes (x, y) ranging from $2 \leq x, y \leq 5$ are tried, making 16 candidate models in total. The best-performing filter sizes for the action models are shown in TABLE V. Further details are decribed in Section VII.

VI. REINFORCEMENT LEARNING

For RL training of Meowjong, we adopt the Monte Carlo policy gradient method: at each time step $t = 0, 1, 2, \cdots, T$ of a round of Sanma, our agent receives a representation of the game state S_t, and on that basis selects an action A_t. One time step later, as a consequence of its action, the agent receives a numerical reward R_{t+1} and finds itself in a new state S_{t+1}. The agent's goal is to maximize the cumulative reward G_t:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2}$$

In a round of Sanma, the reward for each action is not immediately received; instead, the cumulative reward is received once at the end of the round, as the score of the agent, with the intermediate rewards being zero. Therefore, we need a way to estimate the reward for each action, and we introduce a discount rate $0 \leq \gamma \leq 1$ to trade off the immediate and delayed reward: if $\gamma = 0$, the agent is only concerned with maximizing the immediate rewards; as γ approaches to 1, the cumulative reward takes future rewards into account more strongly, making the agent more farsighted.

A policy π is a mapping from each states to a probability distribution over actions, and defines the behaviour of the agent. Here, we use the pre-trained discard CNN as a parametrized policy to be improved through RL:

$$\pi(a|s, \mathbf{w}) = \Pr(A_t = a | S_t = s, \mathbf{w}_t = \mathbf{w}) \tag{3}$$

where $\mathbf{w}$ denotes the weights of the CNN. The state-value function $v_\pi(s)$ and the action-value function $q_\pi(s, a)$ for the policy can then be defined as follows:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \tag{4}$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{5}$$

Algorithm 1 REINFORCE: Monte-Carlo Policy Gradient

Require: Policy parametrization $\pi(a|s, \mathbf{w})$
Require: Learning rate $\eta > 0$
Require: Discount rate $0 \leq \gamma \leq 1$
  Initialize policy parameter $\mathbf{w}$ from the pre-trained model
  while stopping criterion not met **do**
    Generate $S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T$ from $\pi(\cdot|\cdot, \mathbf{w})$
    $R_1, ..., R_{T-1} \leftarrow 0$, $R_T \leftarrow$ Round score/penalty
    $G_T \leftarrow 0$
    for all time step of the episode $t = T - 1$ downto 0 **do**
      $G_t \leftarrow R_{t+1} + \gamma G_{t+1}$
      $\mathbf{w} \leftarrow \mathbf{w} + \eta \gamma^t G_t \nabla_{\mathbf{w}} \log \pi(A_t|S_t, \mathbf{w})$
    end for
  end while

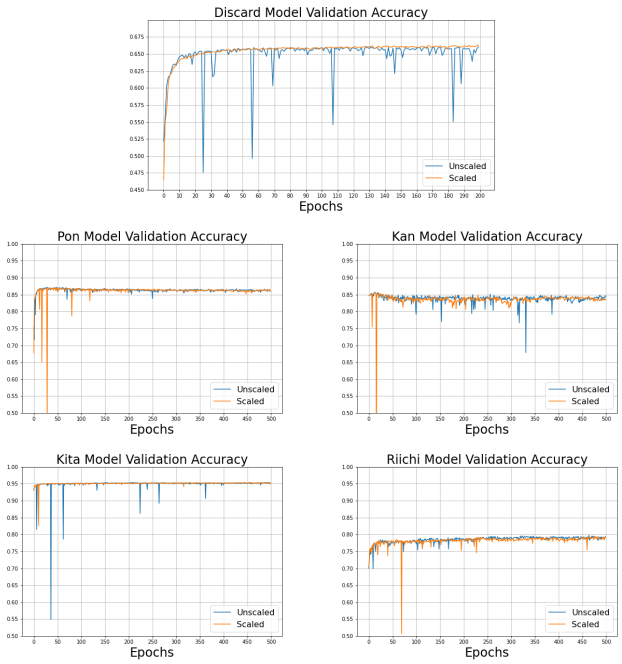

Fig. 3. Learning curves for the action models in supervised learning

We can then define the performance measure $J(\mathbf{w})$ of our agent, given the initial state s_0 of a round of Sanma, and calculate its gradient

$$J(\mathbf{w}) = v_\pi(s_0) \propto \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \pi(a|S_t, \mathbf{w}) \right] \tag{6}$$

$$\begin{aligned} \nabla_{\mathbf{w}} J(\mathbf{w}) &\propto \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \mathbf{w}) q_\pi(S_t, a) \frac{\nabla_{\mathbf{w}} \pi(a|S_t, \mathbf{w})}{\pi(a|S_t, \mathbf{w})} \right] \\ &= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla_{\mathbf{w}} \pi(A_t|S_t, \mathbf{w})}{\pi(A_t|S_t, \mathbf{w})} \right] \\ &= \mathbb{E}_\pi \left[G_t \nabla_{\mathbf{w}} \log \pi(A_t|S_t, \mathbf{w}) \right] \end{aligned} \tag{7}$$

We then use the Monte Carlo policy gradient method to update the weights $\mathbf{w}$ as

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \gamma^t G_t \nabla_{\mathbf{w}} \log \pi(A_t|S_t, \mathbf{w}_t) \tag{8}$$

Algorithm 1 describes how we implement REINFORCE, the Monte Carlo policy gradient algorithm. After hyperparameter tuning, we adopt $\eta = 10^{-3}$ and $\gamma = 0.99$ as the optimum hyperparamter setting. Further details of hyperparameter tuning are decribed in Section VII. Since, for each state S_t, an action A_t is sampled at random from the distribution defined by $\pi(A_t|S_t, \mathbf{w})$, this does introduce the risk of illegal discards: while $\arg\max_{A_t} \pi(A_t|S_t, \mathbf{w})$, which is used for actual predictions, has indeed always been in a player's private hand, the random sampling from $\pi(A_t|S_t, \mathbf{w})$ may still generate a choice that does not exist in the hand. Fortunately, this problem is very rare, occurring only twice during the entire training, and can be bypassed by simply skipping the problematic seeds.

TABLE VI
SIZES OF THE DATASETS

Action	Dataset Size		
	Training	*Validation*	*Test*
Discard	797,285	88,588	147,444
Pon	151,348	16,817	16,887
Kan	34,319	3,814	3,548
Kita	136,924	15,214	15,498
Riichi	109,804	12,201	11,944

TABLE VII
TEST ACCURACIES FOR THE ACTION MODELS IN SUPERVISED LEARNING

Model	Test Accuracy			
	Meowjong	*Meowjong (scaled)*	*Gao et al. [4]*	*Suphx [5]*
Discard	65.81%	65.54%	68.8%	76.7%
Pon	70.95%	72.10%	88.2%	91.9%
Kan	92.45%	88.92%	—	94.0%
Kita	94.26%	94.44%	—	—
Riichi	62.63%	64.29%	—	85.7%

VII. EXPERIMENTS

A. Supervised Learning

We sample 50,000 rounds of Sanma game records from 2019 from the "Houou" ("phoenix" in Japanese) table on Tenhou [2] for training our models. 10% of those data were divided, with stratification, to form the validation datasets of the actions. The "Houou" table is only open for the top 0.1% of the ranked players, so its game records can be considered to be of high quality. To test the generalizability of our models, we sample another 5,000 rounds from 2020 to form the test dataset. We use data from 2020, rather than 2019, as this potentially represents a harder generalization problem. The sizes of the datasets are shown in TABLE VI. We also create a copy of each dataset, with standardizing the data to zero mean and unit variance, to assess its effect on the models' performance. Our models are implemented using TensorFlow, and after hyperparameter tuning, Adam with learning rate 10^{-3} is used as the models' optimizer. The discard models, both with and without data standardization, are trained in mini-batches of size 64 for 200 epochs, and the rest of the models are trained in mini-batches of size 32 for 500 epochs. Each model is trained on 4 NVIDIA P100 GPUs with 64GB memory in total, and takes from 10 hours (Kan model) to 30 hours (discard model).

The learning curves of our models, showing mean validation accuracy can be seen in Fig. 3. They show a uniformly fast and satisfactory convergence. Compared to the discard models, the other models converged much more quickly, which is likely to be due to smaller dataset sizes. According to the learning curves, the scaled (data standardized) and unscaled models show no notable difference. Though one model may converge more stably than the other in some of the actions, this is not a general characteristic across all actions for either model.

TABLE VII shows the test accuracies of Meowjong's models, along with those achieved by previous works. Note that these test accuracies are not directly comparable due to the different training/validation/test data sources and structures,

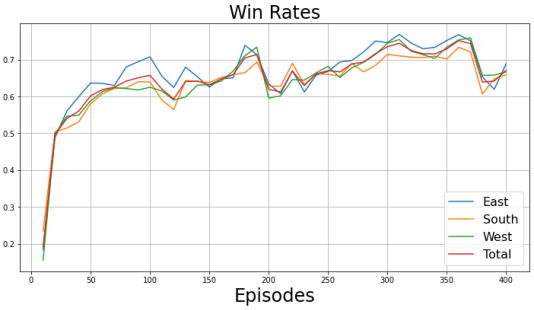

Fig. 4. Win rates of Meowjong in RL training

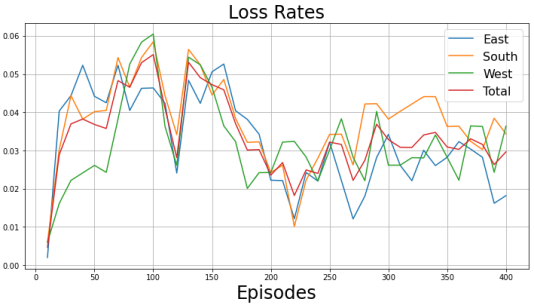

Fig. 5. Loss rates of the RL agent during training

but they can serve as a rough reference. There is still no notable difference in the test accuracies between the unscaled and scaled models. In general, the test accuracies of Meowjong are very satisfactory, achieving Gao et al.'s level [4] at the most important action—discard. Although there is still a gap between Meowjong and Suphx, it is worth pointing out that Suphx used very large, 102/104-layer residual CNN structures, with training datasets of 4M–15M examples, and cost much more in computational power and time to train [5], whereas Meowjong adopted a much simpler CNN structure, used a much smaller training dataset, and was much cheaper and faster to train.

B. Reinforcement Learning

The discard model without data standardization is improved through self-play RL, using the REINFORCE (Monte Carlo policy gradient) algorithm, for 400 episodes. Evaluations against 2 baseline agents (described in Section VIII-A) are carried out and recorded after every 10 episodes, each playing 500 rounds as East, South, and West. The curves of the win rates are plotted in Fig. 4, showing a very successful improvement on all winds. The sudden drops on the win rates between Episodes 190/200 and Episodes 370/380 are probably due to the variance in policy updates, which are performed after every episode.

Fig. 5 shows the loss rates, which also gradually decrease over episodes, after sharp increases in the first 100 episodes. This suggests that the RL agent is likely to have learned an aggressive style, and finessed its skill through self-play, increasing its win rate while lowering its loss rate.

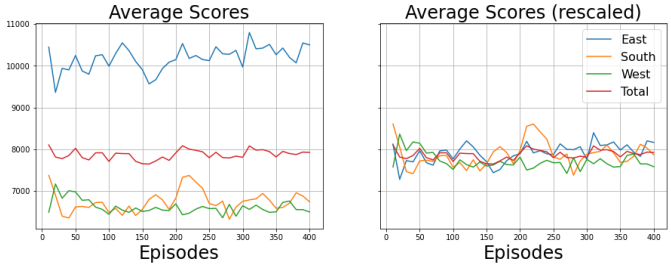

Fig. 6. Average scores of the RL agent during training

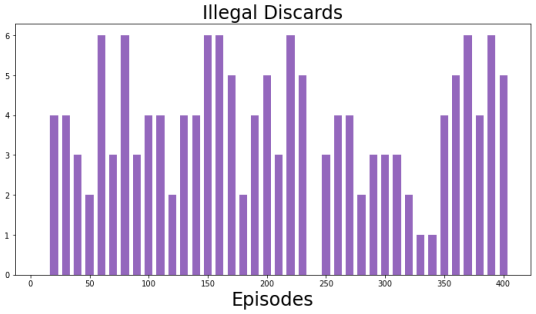

Fig. 7. Illegal discards of the RL agent

Fig. 6 shows the average scores during RL training. The rescaled plots on the right remove the effect caused by the rule that the East player (the dealer) wins 50% more, showing a consistent performance with no preference on any wind. The average scores are also stable across episodes, which is in accordance with the training target for Meowjong—maximising the winning probabilities rather than the round scores.

However, the RL training introduced a side-effect: the discard model may predict a tile to discard that does not exist in the hand. However, this is still very rare in evaluations, occurring at most 6 times in 1,500 rounds, and can be avoided in the deployment phase by predicting the legal tile with the highest probability. According to the bar plot in Figure 7, we believe that the rate of illegal discards is independent of the episode.

VIII. Agent Evaluation

A. Agents

We train the following agents for evaluation:

- SL agent: the supervised learning agent with all 5 models trained using supervised learning;
- SL-scaled agent: the supervised learning agent using the data-standardized models;
- RL agent: the reinforcement learning agent with the Pon, Kan, Kita and Riichi models inherited from the SL agent, but its discard model initialized as the SL discard model and enhanced through RL.

We also build an agent that takes actions randomly to serve as a baseline. This agent is believed to have a similar power to the bots on the major online platforms.

TABLE VIII
COMPARISONS BETWEEN MEOWJONG AGENTS VS. BASELINE AGENTS

Agents (vs. Baseline)	Wind	1st Place Rate	2nd Place Rate	3rd Place Rate	Draw Rate
Baseline	—	0.02%	0.02%	0.02%	99.94%
SL	East	22.00%	0.06%	0.08%	77.86%
	South	22.68%	0.06%	0.02%	77.24%
	West	20.72%	0.16%	0.04%	79.08%
	Total	21.80%	0.09%	0.05%	78.06%
SL-scaled	East	21.40%	0.04%	0.14%	78.42%
	South	22.72%	0.08%	0.00%	77.20%
	West	20.02%	0.20%	0.06%	79.72%
	Total	21.38%	0.11%	0.06%	78.45%
RL	East	73.59%	0.02%	3.27%	23.12%
	South	71.93%	0.08%	3.46%	24.53%
	West	71.61%	0.06%	2.85%	25.48%
	Total	72.38%	0.05%	3.19%	24.38%

TABLE IX
SIGNIFICANCE TEST RESULTS FOR MEOWJONG VS. BASELINE AGENTS

Agents (vs. Baseline)	Wind	n	x_1	x_2	x_3	x_{draw}	$\Pr(\mathbf{X} \succeq \mathbf{x})$
SL	East	5,000	1,100	3	4	3,893	$< O(\epsilon)$[a]
	South	5,000	1,134	3	1	3,862	$< O(\epsilon)$
	West	5,000	1,036	8	2	3,954	$< O(\epsilon)$
	Total	15,000	3,270	14	7	11,709	$< O(\epsilon)$
SL-scaled	East	5,000	1,070	2	7	3,921	$< O(\epsilon)$
	South	5,000	1,136	4	0	3,860	$< O(\epsilon)$
	West	5,000	1,001	10	3	3,986	$< O(\epsilon)$
	Total	15,000	3,207	16	10	11,767	$< O(\epsilon)$
RL	East	4,949	3,642	1	162	1,144	$< O(\epsilon)$
	South	4,949	3,560	4	171	1,214	$< O(\epsilon)$
	West	4,949	3,544	3	141	1,261	$< O(\epsilon)$
	Total	14,847	10,746	8	474	3,619	$< O(\epsilon)$

[a] ϵ is the smallest representable positive `float64` value in Python, which equals $2^{-1074} \approx 4.94 \times 10^{-324}$, and the constant factor is at most 15000^3, so the cumulative probability cannot exceed 1.67×10^{-311}.

B. Evaluation Metrics

The initial private tiles have large randomness and will greatly affect the win/loss of a game. In order to test whether one agent performs significantly better than another, we let the agents play a substantial number of rounds against each other, and conduct appropriate significance tests on the results. Since a round of Sanma can have 4 different outcomes: 1st place, 2nd place, 3rd place and draw, we assume that the game results amongst 3 agents can be modelled by a multinomial distribution Multinomial$(n, \mathbf{p})$, where:

- n is the total number of rounds played;
- $\mathbf{p} = \begin{bmatrix} p_1 & p_2 & p_3 & p_{\text{draw}} \end{bmatrix}$ denotes the probabilities of each outcome, with 1st place rate p_1, 2nd place rate p_2, 3rd place rate p_3, and draw rate p_{draw}.

The probability mass function for Multinomial$(n, \mathbf{p})$ is:

$$\Pr(\mathbf{X} = \mathbf{x}) = \frac{n!}{x_1!x_2!x_3!x_{\text{draw}}!} p_1^{x_1} p_2^{x_2} p_3^{x_3} p_{\text{draw}}^{x_{\text{draw}}} \tag{9}$$

where $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_{\text{draw}} \end{bmatrix}$ denotes the frequencies of the outcome, with 1st place frequency being x_1, 2nd place frequency being x_2, 3rd place frequency being x_3, and draw frequency being x_{draw}.

Before conducting the significance testing, it is necessary to define what is "better". Here we define "better" to be having both a higher 1st place rate and a lower 3rd place rate. If both the 1st place rates and the 3rd place rates are equal, the agent should prioritize on improving the draw rate, because 2nd place often comes from another player winning by self-draw or from the third player's discard, resulting in the agent themself having a zero or negative round score, and a guaranteed negative score difference. Two outcomes with one having higher values on both the 1st place rate and the 3rd place rate, or lower values on both the 1st place rate and the 3rd place rate, are not directly comparable. Therefore, for two outcomes $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_{\text{draw}} \end{bmatrix}$ and $\mathbf{x}' = \begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_{\text{draw}} \end{bmatrix}$, we define the following strict partial order relation $\succ$, interpreted as "an outcome is better than another", to be

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}'.\ \mathbf{x} \succ \mathbf{x}' \iff & (x_1 > x'_1 \wedge x_3 < x'_3) \vee \\ & (x_1 = x'_1 \wedge x_3 = x'_3 \wedge x_{\text{draw}} > x'_{\text{draw}}) \end{aligned} \tag{10}$$

The same "better" relation also applies to the probabilities. Therefore, with the baseline probabilities $\mathbf{p}_0 = \begin{bmatrix} p_1 & p_2 & p_3 & p_{\text{draw}} \end{bmatrix}$ observed from the baseline game results, we set up the following hypotheses:

Null hypothesis H_0: $\mathbf{p} = \mathbf{p}_0$
Alternative hypothesis H_a: $\mathbf{p} \succ \mathbf{p_0}$
Significance level: $\alpha = 0.05$

The cumulative distribution function of Multinomial$(n, \mathbf{p})$ can be used to test H_0, which can be calculated by the following formula:

$$\begin{aligned} \Pr(\mathbf{X} \succeq \mathbf{x}) &= \sum_{\mathbf{x}' \succeq \mathbf{x}} \Pr(\mathbf{X} = \mathbf{x}') \\ &= \sum_{x'_1 > x_1} \sum_{x'_2} \sum_{x'_3 < x_3} \Pr\left(\mathbf{X} = [x'_1\ x'_2\ x'_3\ (n - x'_1 - x'_2 - x'_3)]\right) \\ &\quad + \sum_{x'_2 < x_2} \Pr\left(\mathbf{X} = [x_1\ x'_2\ x_3\ (n - x_1 - x'_2 - x_3)]\right) \\ &\quad + \Pr\left(\mathbf{X} = [x_1\ x_2\ x_3\ x_{\text{draw}}]\right) \end{aligned} \tag{11}$$

TABLE X
COMPARISONS BETWEEN SL AND RL AGENTS

Agents	Wind	1st Place Rate	2nd Place Rate	3rd Place Rate	Draw Rate
SL vs. SL	East	18.70%	11.90%	24.84%	44.56%
	South	19.74%	24.32%	11.38%	44.56%
	West	17.00%	19.22%	19.22%	44.56%
	Total	18.48%	18.48%	18.48%	44.56%
SL vs. RL	East	5.76%	7.09%	81.53%	5.62%
	South	6.10%	38.09%	49.36%	6.45%
	West	4.90%	37.69%	51.68%	5.72%
	Total	5.59%	27.62%	60.86%	5.93%
RL vs. SL	East	57.46%	7.75%	13.92%	20.87%
	South	57.90%	16.34%	4.93%	20.83%
	West	55.68%	18.18%	5.25%	20.89%
	Total	57.02%	14.09%	8.03%	20.86%

TABLE XI
SIGNIFICANCE TEST RESULTS FOR RL VS. SL AGENTS

Agents	Wind	n	x_1	x_2	x_3	x_{draw}	$\Pr(\mathbf{X} \succeq \mathbf{x})$
RL vs. SL	East	4,993	2,869	387	695	1,042	$< O(\epsilon)$
	South	4,993	2,891	816	246	1,040	$< O(\epsilon)$
	West	4,993	2,780	908	262	1,043	$< O(\epsilon)$
	Total	14,979	8,540	2,111	1,203	3,125	$< O(\epsilon)$
SL vs. RL	East	4,996	288	354	4,073	281	$> 1 - O(\epsilon)$
	South	4,996	305	1,903	2,466	322	$> 1 - O(\epsilon)$
	West	4,996	245	1,883	2,582	286	$> 1 - O(\epsilon)$
	Total	14,988	838	4,140	9,121	889	$> 1 - O(\epsilon)$

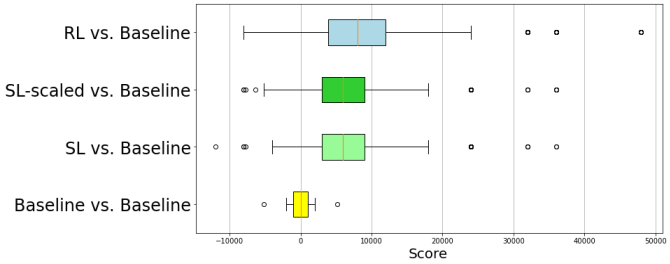

Fig. 8. Scores of Meowjong agents against baseline agents

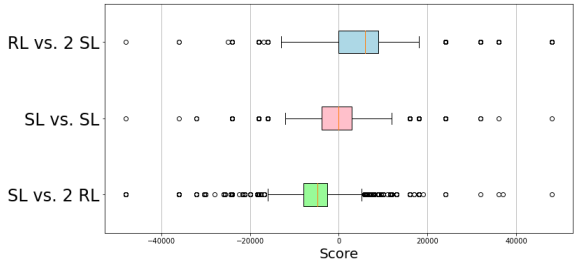

Fig. 9. Scores of the SL agents vs. RL agents

where $\succeq$ is the non-strict version of $\succ$. If $\Pr(\mathbf{X} \succeq \mathbf{x}) < \alpha$, then H_0 can be rejected.

C. Evaluation Results

We simulate 5,000 rounds of Sanma games amongst 3 baseline agents, and 5,000 rounds for each of the SL, SL-scaled, and RL agents against 2 baseline agents, in each wind position. Their results, in terms of 1st/2nd/3rd-place and draw rates, are reported in TABLE VIII. The results clearly show that all Meowjong's trained agents can outperform a baseline agent. There is no significant difference in any of the rates between the SL and SL-scaled agents, and the RL agent's 1st place rates are also much larger than both SL agents. The RL agent also has higher 3rd-place rates, which suggests that it has learned an aggressive style through self-play.

The results of the significance test are reported in TABLE IX, which reject all H_0 and prove the significant difference. This is also confirmed by the box plots of all the scores shown in Fig. 8. Besides, it took on average 4.2 hours to simulate 5,000 rounds, which is about 3 seconds per round. This means Meowjong is able to make decisions very quickly, satisfying the time limit of all online platforms.

D. Comparison Between SL and RL Agents

In this series of evaluations, we simulated 5,000 rounds amongst 3 SL agents, and 5,000 rounds in each wind positions in both SL vs. 2RL and RL vs. 2SL. The results are reported in TABLE X. It is clear from the outcomes that an RL agent can outperform an SL agent. The higher 3rd place rates at the East position are due to the rule that the East player pays twice as much as the other player when a non-East player wins by self-draw. The significant difference can also be confirmed by the significance test results in TABLE XI, and the box plots of the scores in the simulation in Fig. 9. Besides, according to the outcomes, both SL agents seem to perform better at the South position.

IX. Conclusions

In this paper, we design a new data structure to encode the observable states in Sanma games, build an efficient CNN structure that solves the Sanma's decision-making problem, and train several Sanma agents using both supervised learning and reinforcement learning. All our action models achieve test accuracies comparable with AIs for 4-player Mahjong through supervised learning, and gain a significant further enhancement from reinforcement learning. Being the first ever AI in Sanma, we can claim that Meowjong stands as a state-of-the-art in this game.

In future work, we plan to take the multi-round ranking information into account, and let Meowjong try to maximize the full-game results, in addition to the single-round performance. An expert player adapts to different strategies flexibly based not only on their score and ranking at the beginning of each round, but also the progress of the entire game. For example, a player may play defensively in the later rounds when they possess a great lead, and may even lose deliberately to the lowest-placed player by a small amount in the last round, in order to secure the 1st place. If Meowjong could learn such a flexible full-game strategy, this would surely improve its multi-round performance even further. The nature of our data structure for encoding the game state already supports such an upgrade, and sequential neural network structures, such as GRUs and long short-term memory networks (LSTMs), might have the best potential for tackling this task.

Acknowledgment

This work has been performed using resources provided by the Cambridge Tier-2 system operated by the University of Cambridge Research Computing Service (www.hpc.cam.ac.uk) funded by the Engineering and Physical Sciences Research Council Tier-2 capital grant [EP/P020259/1].

References

[1] N. Mizukami and Y. Tsuruoka, "Building a computer Mahjong player based on Monte Carlo simulation and opponent models," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2015, pp. 275–283.

[2] S. Tsunoda, "Tenhou.net," https://tenhou.net/.

[3] M. Kurita and K. Hoki, "Method for constructing artificial intelligence player with abstractions to Markov decision processes in multiplayer game of Mahjong," *IEEE Transactions on Games*, vol. 13, no. 1, pp. 99–110, 2021.

[4] S. Gao, F. Okuya, Y. Kawahara, and Y. Tsuruoka, "Supervised learning of imperfect information data in the game of Mahjong via deep convolutional neural networks," *The 23rd Game Programming Workshop of the Information Processing Society of Japan*, 2018.

[5] J. Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T.-Y. Liu, and H.-W. Hon, "Suphx: Mastering Mahjong with deep reinforcement learning," *arXiv preprint arXiv:2003.13590*, 2020.