# Phoenix: An Open-Source, Reproducible and Interpretable Mahjong Agent


Jun Lin
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
junlin@usc.edu

Yu Xing
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
xingy@usc.edu

Jingwen Sun
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
sunjingw@usc.edu

Bingling Huang
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
binglinh@usc.edu

Aiqi Liu
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
aiqiliu@usc.edu

Na Li
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
nli10945@usc.edu

Jiabao He
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
jiabaohe@usc.edu

Chen Cui
*Computer Science Department, Viterbi*
*University of Southern California*
Los Angeles, USA
cuichen@usc.edu



***Abstract*—Mahjong is the multi-round tile based multiplayer game with imperfect-information. It has more complex scoring rules than Go and AI has been shown to achieved great success in the field of perfect-information games. To tackle the issue of imperfect information, We adapt the fundamental idea from Suphx [1] by designing several supervised models with some custom feature engineering as the foundation and later using reinforcement learning technique such as proximal policy optimization [6] with distribute Ray framework[1] to further improve. At the end, we achieved 57% categorical accuracy on discard model and 80%, 84%, 95%, 95% for Chi, Pon, Kan, and Riichi model respectively.**



***Index Terms*—Supervised Learning, Reinforcement Learning, Distribute training**


## I. Introduction

Riichi Mahjong[2] is a board game with four players and is one of the most popular mahjong variants worldwide. It is usually played with 136 tiles. There are 34 different kinds of tiles, with four of each kind. All 34 kinds could be divided into tiles suits of pin (circles), so (bamboo), wan (characters) and unranked honor tiles.

At the beginning of a typical game, each player has a randomized initialized hand consist of 13 private hands. The other tiles are shuffled as the wall (piles). In each round, every player draws a tile from the wall and then discard a tile from his hand. Besides, a player can make a meld (open group) by calling another player's discard for melds. A typical meld could be three or four same tiles or three continuous tiles. Once a player has collected four melds and a pair, he could declared his winning in this round and end this round. Informally, we could describe a winning hand as x * (AAA) + y * (ABC) + DD while x + y = 4, where AAA and ABC represents melds and DD represents the pair.

The goal of project Phoenix is to produce an open-source and interpretable Mahjong Agent which could be used to populate Riichi Mahjong as well as improving people's level in Mahjong.

[1]Ray is a simple, universal API for building distributed applications.
[2]Riichi Mahjong, En.wikipedia.org

## II. Related Work

### A. Mahjong AI

Early Mahjong AI work are mainly focus on using statistic models such as Markov Chain [4] or Monte Carlo Simulation [5]. Recently more and more deep models are utilized in modeling the decision model in Mahjong. Starting from combining deep models with rule-based models to completely deep learning based [2].

Microsoft Research Asia has recently developed a Mahjong AI named Suphx which beat 99.9 percent human players in Tenhou platform [1]. The biggest improvement comparing to previous models are that Microsoft brings Reinforcement Learning, oracle guiding and global reward predictor into modeling. Global reward predictor trains a predictor to predict the final reward of a game based on the information of the current and previous rounds. Oracle guiding introduces an oracle agent that can see the perfect information including the

private tiles of other players and the wall tiles then gradually remove the information.

Microsoft Research Asia evaluates Suphx on Tenhou, which is a web based mahjong platform in Japan with a complete ranking system and over 350,000 users. It shows that Suphx has beaten most of human players and reaches the highest 10 dan.

### B. Reinforcement Learning

The idea of learning from interacting with the environment can be traced to infant plays [1]. Such interactions may become a source of knowledge that can be used and applied to conduct competitive or exploratory tasks.Learning from interactions is a fundamental idea underlying nearly all theories of learning and intelligence. Reinforcement Learning, which is one of them, focuses on goal-oriented learning from interactions than other approaches [7].

Some challenges arose in reinforcement learning that are not in other types of learning methods. The first one is the trade-off between exploration and exploitation [7]. The agent has to exploit and explore in the learning process but they may lead to different influences on rewards. Exploitation means taking action as what has already been experienced. In contrast, exploration means taking actions that were not selected before. The challenge is that neither exploration or exploitation can be pursued exclusively without failing at the task [7]. Even though this problem has been intensively studied by mathematicians for a long time, it still remains unresolved yet [7]. Another challenge of reinforcement learning is that it explicitly considers the whole picture of the task for a goal-oriented agent interacting with an uncertain environment [7]. The dilemma is that it contrasts many approaches that consider dividing a big problem into several sub-problems.

Overall speaking, reinforcement learning starts with interactive and goal-seeking learning agents that have explicit goals and have the ability to perceive the environment. After agents taking actions, the environment is influenced and agents may keep perceiving the environment.

### C. Bakuuchi

Bakuuchi is another player who performs Monte Carlo simulation [4].The probability of Bakuuchi has higher accuracy, the equation. Evaluate each simulation and the simulation strategy learned from the game record at the end of the hand. In an early study [4], they reported that the point dependence on policy was inappropriate and only reached an intermediate level. It is worth noting that Bakuuchi has reached an advanced level. As far as we know, no tree has been found to find better decisions.

### D. NAGA

Mahjong is a difficult game because you can't see your opponent's hand and don't know which card you will draw next. Mahjong also requires a long-term strategy, and it is difficult to predict the state of the next round. Traditional Mahjong AI deals with these problems based on the expected final ranking and Monte Carlo method, while NAGA solves these problems through deep learning[5].

Naga uses confidence to learn each CNN[5], and if you are not confident in the distribution of predicted behavior, you can refer to the correct answer by penalizing it. Therefore, the phase where the confidence is low is that NAGA can not have confidence in the tile selection, that is, it is a lost phase.

Naga enters the table information and calculates the slope that can expand the selection range[5], so that the produced features can be visualized. Conversely, you can also visualize features that will no longer make a selection by calculating the slope of the selection attenuation. In the deepest layer of the neural network, it seems that judgments are made with complex granularity.

## III. Dataset

### A. Data Sources

Thanks to the well designed logging system provided by Tenhou platform, each game is logged in the XML format (Extensible markup language). Player's actions and table information are recorded as XML tags. In order to bootstrap the process of supervised learning training, we scrape the logs bundle from the Tenhou platform server. Each file is bundled together by year since 2009. Although earlier logs from 2005 exist, but the highest ranking game logs are available only after 2009. Since we are targeting for high quality games, the low ranking games and three-player game logs are filtered. Here are statistics of our datasets. For some unknown reason the logs in 2020 are missing in the Tenhou platform. We will skip them for now.

TABLE I: Size of Raw Data

| 2009 | 2010 | 2011 | 2012 |
|---|---|---|---|
| 51k | 40k | 464k | 85k |
| 2013 | 2014 | 2015 | 2016 |
| 71k | 48k | 46k | 44k |
| 2017 | 2018 | 2019 | 2021 |
| 30k | 36k | 46k | 1332 |

### B. Data Preprocessing

*1) Data for Supervised Learning:* In order to generate training data for supervised learning, we need to take a snapshot of the table before each actions. Since raw data is a log file which means it increases incrementally. We have to simulate the changes according to the log and take snapshots to our table.

Firstly, we will use the log parser to resolve data from logs, and extract features from it. We extracted features for the Discard Model, Riichi Model, Chi Model, Pon Model, and Kan Model. In this step, we generate a 2d vector for each state as the input of a supervised learning neural network. The detailed format will be shown as below. In this part, we also made an effort to remove three-player log, and fixed logs with duplicate nodes. Table II shows the distributions of classes that each action performed after the preprocessing.

TABLE II: Classes Distribution

| | Chi | Pon | Kan | Riichi |
|---|---|---|---|---|
| No | 15,666,962 (0.88) | 3,155,855 (0.97) | 1,117,335 (0.84) | 2,697,554 (0.59) |
| Yes | 1,972,332 (0.12) | 92,890 (0.03) | 202,184 (0.16) | 1,843,236 (0.41) |
| Total | 17,639,294 | 3,248,745 | 1,319,519 | 4,540,790 |

*2) Data for Real-time Gaming:* Features are needed as an input for trained models in real-time gaming. So generating exactly the same features as the training data from tenhou bot interfaces, which is an access to play riichi mahjong on tenhou.net server, is necessary.

### C. Data Format

*1) States Format:* We have different data formats in different models. The following are the data formats for the Discard Model and Riichi_Chi_Pon_Kan Model. Both of them contains the metadata of the current round, and player's tile details, and enemies tile details. All of these factors may affect the player's decision.

| Attributes | Value Type or Options |
|---|---|
| player_id | 0/1/2/3 |
| draw_tile | [action, 136tiles] |
| player_tiles | closed hand: [136tiles]<br>open hand: [136tiles]<br>discarded tiles: [136tiles] |
| open_hands_detail | tiles: [136tiles]<br>meld_type: Chi/Pon/AnKan/MinKan/KaKan<br>reacted_tile: [136tiles] |
| enemies_tiles | enemy_seat: 0/1/2/3<br>closed_hand: [136tiles]<br>open_hand: [136tiles]<br>discarded_tiles: [136tiles] |
| dealer | 0/1/2/3 |
| repeat_dealer | int |
| riichi_bets | int |
| player_wind | S/N/E/W |
| prevailing_wind | S/N/E/W |
| dora | [136tiles] |
| scores | [scores for player seat 0, 1, 2, 3] |
| discarded_tile | [136tiles] |

Fig. 1: Data Format for Discard Model

*2) Extracted Feature Format:* General feature is a (D, 34) vector include information such as player tiles(12,34), enemies tiles(36,34), dora(5,34), current scores(4,34), and other basic board information(5,34) like dealer and wind. Manual crafted features will also be concatenated. Each model has its own individual feature, for example, tile to Chi is an individual feature important to Chi feature generator.Thus the final feature will be individual feature concatenate to general feature.

## IV. Methodologies

### A. Base Model

Our supervised models take the ideas from the well known DenseNet [3] architecture. DenseNet embrace the idea of short-cut connection from ResNet and connects each layer to every other layer in a feed-forward fashion. There are several

| Attributes | Value Type or Options |
|---|---|
| player_id | 0/1/2/3 |
| dealer | 0/1/2/3 |
| repeat_dealer | int |
| riichi_bets | int |
| player_wind | S/N/E/W |
| prevailing_wind | S/N/E/W |
| player_tiles | closed hand: [136tiles]<br>open hand: [136tiles]<br>discarded tiles: [136tiles] |
| open_hands_detail | tiles: [136tiles]<br>meld_type: Chi/Pon/AnKan/MinKan/KaKan<br>reacted_tile: [136tiles] |
| enemies_tiles | enemy_seat: 0/1/2/3<br>closed_hand: [136tiles]<br>open_hand: [136tiles]<br>discarded_tiles: [136tiles] |
| dora | [136tiles] |
| scores | [scores for player seat 0, 1, 2, 3] |
| last_player_discarded_tile | [136tiles] |
| could_chi | 0 for false, 1 for true |
| could_pon | 0 for false, 1 for true |
| could_minkan | 0 for false, 1 for true |
| is_FCH | 0 for false, 1 for true |
| action | [136tiles] |

Fig. 2: Data Format for Riichi/Chi/Pon/Kan Model

advantage of using DenseNet architecture, it has a significantly small parameter size and strengthened feature propagation. The implementation of DenseNet [3] consist four identical dense blocks. Each block consists of one convolutional layer with kernel size 1 x 1 and another convolutional layer with 3x3 kernel size. The difference between each block is the number of such pairs in the block. In our case, we used in total 201 layers. First block has the 6 such pairs, the second block with 12 pairs, the third block has 48 pairs, and the last block with 32 pairs. After each block, there are one more 1x1 convolutional layer and one 2x2 average pooling layer. Note that instead of performing the skip connection in ResNet, every layer in the block directly connects to all subsequent layers. At the end, we added 2 more dense layers to smooth down the distribution and softmax activation function to distribute the probabilities to 34 classes for Discard model and binary classes for Chi, Pon, Kan, and Riichi model. Since discard and other models has different number of classes and they are also evaluate differently. In RCPK models, binary cross entropy is employed and start with 1e-4 learning rate and slowly decay after few epoch. Discard also use the same optimizer but with categorical cross entropy for better evaluation on the performance. Besides of common metric such as accuracy and are under the curve (AUC), we also plot out the confusion matrix for better understand of our RCPK model prediction.

### B. Reinforcement Learning in Mahjong

The complexity of mahjong is two fold. The first complexity is that we take different strategies under different circumstances. Most of the time the following factors are considered. Firstly, the level of your enemies. If in a low level room, you could defend less and become more aggressive. Secondly, our

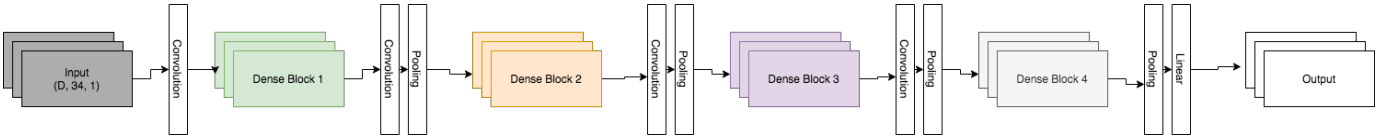


Fig. 3: Discard, Chi, Pon, Kan, and Riichi Model

current rank in this game. If we are currently rank 1, we tend to be conservative to avoid dealing in. Otherwise, we tend to play aggressively to gain more scores. Thirdly, the quality of initial hand. With low-quality initial hand, we will have more difficulty reaching tempai. In this process, we are prone to dealing in leading worse status.

The second complexity is that how we invade or defend. Mahjong is an imperfect information game and the randomness bring more challenge for players to make a plan to invade or defend. A common strategy is to determine which tile is safe based on others' discarded tiles. There are many mahjong research about the correlation between safe tile and others' discarded tiles.

Here we mainly take the methods from Suphx. To solve the challenge of imperfect information, oracle guiding is applied. Firstly a strong agent is trained by providing all information, both visible and invisible. This agent will perform really well because he could see others' private tiles and take the best strategy. Afterwards we gradually decay the invisible information in the training. We will end up with an agent with visible information only. We could say that the visible information are now utilized appropriately in their knowledge structure.

As for the challenge of strategies, we take the same global reward predictor as Suphx. The advantages of global reward predictor is two fold. Firstly, it could help us distribute the final rewards into each round in a game so that we could get more precise and instant reward after a round. Secondly, it should be mentioned that losing scores doesn't always mean negative signals. To keep our advantageous state, if we start with a low-quality hand, we should decidedly defend to avoid lose too many scores. Low quality initial hand means high risk and low gains if blindly try to win. The global reward predictor will impose positive signals to "reasonable losing" and big penalty to "obtrusive losing".

An asynchronous and distributed learning method is applied. The RL pipeline could be divided into client side and server side. The client side integrates our agents into Tenhou platform and collect state, actions and rewards into replay buffer. The parameter servers keep training latest deep models using recent logs in replay buffer using policy gradient. The client would fetch latest parameters from server side after a certain period.

After the game information is saved in the replay buffer, it can be referred to by other parts when needed. As for the reinforcement learning algorithm, it needs to know current state, action, reward and current parameters as input and updates policy by training the neural network's parameters separately for five models which are riichi, chi, peng, kong

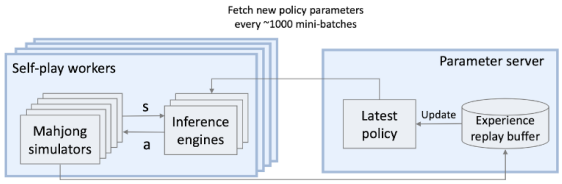


Fig. 4: Reinforcement Learning Structure in Mahjong Training

and discard. By updating the new neural network, the system can select action with the biggest probability at current state. And the updated policy and action will be returned to the Mahjong environment. That is how it works in the system.

Fig. 5: Online Training Environment

When we consider choosing RL algorithms, there are two constraints. The first point is, compared to common games which give instant reward for each step, mahjong is more similar to board game Go. You could only get feedback after completing each round. The second point is that, in mahjong it is hard to build a network to estimate the Q-value of the current board situation. Unlike Go, the advantage and disadvantage didn't show on board. Player should decide on attack or defend based on the quality of his own hands as well as deduce what would be a dangerous tile based on others' discarded tiles. Based on these two considerations, we choose policy gradient as our RL algorithms. And the objective function is shown as below [1].

$$\Delta_\theta J(\pi_\theta) = \underset{s,a\sim\pi_{\theta'}}{E}\left[\frac{\pi_\theta(s,a)}{\pi_{\theta'}(s,a)}\Delta_\theta log\pi_\theta(a|s)A^{\pi_\theta}(s,a)s\right] + \alpha\Delta_\theta H(\pi_\theta) \quad (1)$$

As mentioned before, we can not get instant feedback in each timestep. We will use processed round results (will mention below) as Advantage.

We could get round scores after each round. However, as we mentioned above, the gains and loss of score in each round didn't absolutely mean good or bad feedback for your policy. Take a simple example, you start with a bad initial hand so you take a conservative method and avoid to lose more scores by 'Ron'. Finally you lose a little points because others made a 'Tsumo'. In another round, you have a good initial hand and take an aggressive method. You won a lot of scores in this round and finally got rank 1. In this example, we should give positive signals for both rounds. That's why we need the global reward predictor. In our system, the global reward predictor $\Phi$ is a recurrent neural network and is trained by minimizing the following mean square error [1]:

$$min\frac{1}{N}\sum_{i=1}^{N}\frac{1}{K_i}\sum_{j=1}^{k_i}(\Phi(x_i^1,\ldots,x_i^j)-R_i)^2 \quad (2)$$

Where N is the number of games in the training data, $R_i$is the final game reward of the i-th game, $K_i$is the number of rounds in the i-th game. $x_i^k$ the feature vector of the k-th round in the i-th game, including the initial score of current round the gains of this round and some other round features such as riichi bets. In the online pipeline, the global reward predictor will be given a history round feature after each round and will give the reward for current round. The reward will be used in advantage computation and stored in reply buffer.

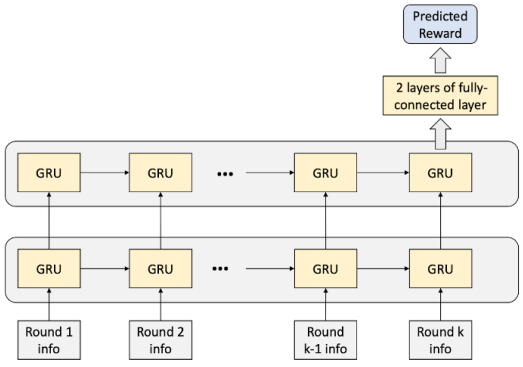


Fig. 6: Global Reward Predictor Work Flow

### C. Distributed Reinforcement Learning

We used the Ray framework developed by RISE Lib to build our distributed training pipeline. Here are five components.

- Replay Buffer (distributed): Store experiences.

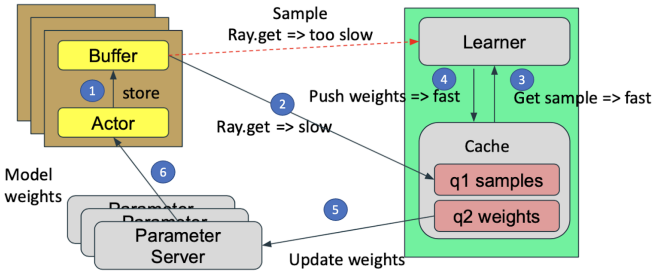


Fig. 7: Distributed Reinforcement Learning Pipeline

- Parameter Server (distributed): Store most updated model weights.
- Actor (distributed): Self-play or play with real users, responsible for collecting experiences.
- Learner (centralized): Pull samples from cache, train and push weights to cache.
- Cache (centralized): Built inside Learner, use multiprocessing to communicate with replay buffer and parameter server. There are two multiprocessing queues inside cache. The first is to store samples, the second is to store model weights.

## V. Experiments

### A. Google Cloud Platform

*1) Preprocessing using Dataflow:* Processing such a huge dataset locally is infeasible and impractical. Since Google Cloud Platform(GCP) offers $300 free credits for all new users, we have decided to use GCP to do preprocessing and training. In order to work on these dataset more efficiently, we move all our dataset from local storage to the google cloud storage bucket, this will significantly speed up the disk I/O and lower the network latency for other GCP services.

Utilizing the preprocessing logic from the III-B, we employ an Apache Beam pipeline to speedup and scale for processing our dataset. Apache beam is a unified programming model to define and execute data processing pipelines, including extract, transform, and load(ETL), batch and stream (continuous) processing. Beam can easily do distributed processing on the fly and the GCP dataflow services use Apache Beam as the main backbone.

In the preprocessing stage, the input of the pipeline is directly sourced from the google storage bucket. Logic in the pipeline is defined by one unit, meaning that every operation logic is done by one unit. Only one row of data is processed at a time. This is for the purpose of multithreading and multi workers distribute processing. Since each row in a csv file is a game log, we read the csv files line by line. In the next step of the pipeline, we convert the raw text to csv row data and only the log column is extracted. The extracted column consists of only one log in the raw XML format. We need to parse the XML format to an easy to manage format. This is done in the III-B. After getting the result from III-B in dictionary form, we transform it to the shape of (D,34,1), the first dimension is

determined by different features in different models. Filtering out some of the invalid data points, we split it into train and validation datasets for later training. These datasets are saved into the TFRecord files in the bucket, we will use them to load our dataset into Tensorflow training. At the point, we have concluded the process of preprocessing, all the relevant file are generated and ready to begin the training process.

*2) Distribute training using Mirrored Strategy:* Training on a single GPU is slow and impractical in bigdata era. With the growth of cloud services providers like Amazon web servises(AWS) and Google Cloud Platform (GCP), distribute training is not longer the headache for researchers. Especially, with the help of higher level machine learning frameworks like Keras and Tensorflow, researchers can deploy multi-GPUs or even multi worker with multi-GPUs with ease. In our setting, we expedite the process by utilized the Mirrored Strategy with the Tensorflow Api. Mirrored Strategy is a synchronous training strategy that works in a single machine with multiple GPUs. In GCP, we are allowed to attach some amount of GPUs to a server. This is some fast way to train with multiple GPUs without any overhead. The main idea of Mirrored Strategy is to make a copy of the model variable to all the processors. During the training, it will combine the gradients all together and apply changes to all copies of processors. There are some other kinds of strategies. We currently adopt Mirrored Strategy due to the simple usage and the more stable compare to others. Since most of our training code is written in Keras and Keras is the higher level API for tensorflow, we can easily configure the strategy on our existing code without much modification. All we need to do is to add the appropriate scope to our model and it will do the rest for us. In our experiment, we used a master server with n1-highmem-8 machine which has 8 virtual CPU (vCPU) and about 7 GB memory per vCPU. We attached 4 Nvidia T4 GPUs to our master server for our training task. With the help of multi GPUs, we are able to quickly train our RCPK model in few hours. These models have smaller datasets compared to discard models due to the nature of these actions. On the other hand, train the discard model is much slower compared to RCPK models and more difficult to get great performance out of the box.

### *B. Training results with Supervised models*

As the foundation of our mahjong system, we want these supervised models to have some comparable level of skills to human players, so we can continue to improve using the reinforcement learning techniques later. In Fig 8, discard model achieved with about 60% categorical accuracy for this task. Although the result seem to be low, but in the example run, the performance of discard model is notable against human player. Compare to the previous work from Suphex [1], we are achieving some comparable results without any tuning. For the Kan and Pon models in Fig 8, 9, we slightly outperform the baseline set by the Suphex [1]. There are still some room for improvement with parameter tuning due to the time constraint, we will skip it for now.

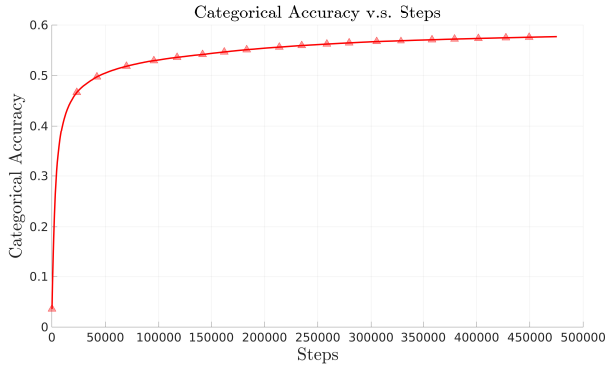


Fig. 8: Discard model accuracy

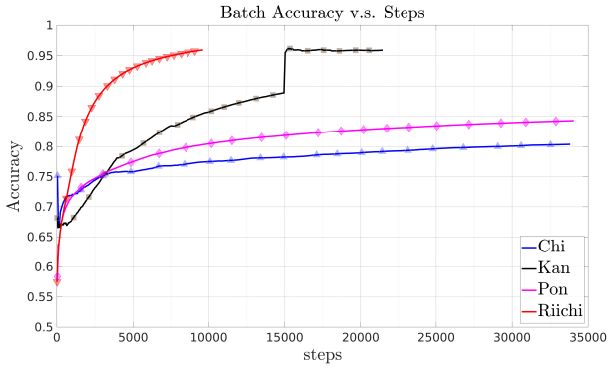


Fig. 9: Chi, Pon, Kan, and Riichi model accuracy

### *C. Tenhou Platform Integration*

One important aspect of our project is the ability to play mahjong in real-time with human players. In the other hand, the reinforcement learning needs an environment for agents to play around as well. In our current setup, agents are allow to play with builtin computer player, with other three human players, or form an table privately and play. We utilized the the builtin computer players in our debugging phase. Once everything work in the environment with builtin computer players, We release our agent to the public available table in tenhou to evaluate the performance against human player. In

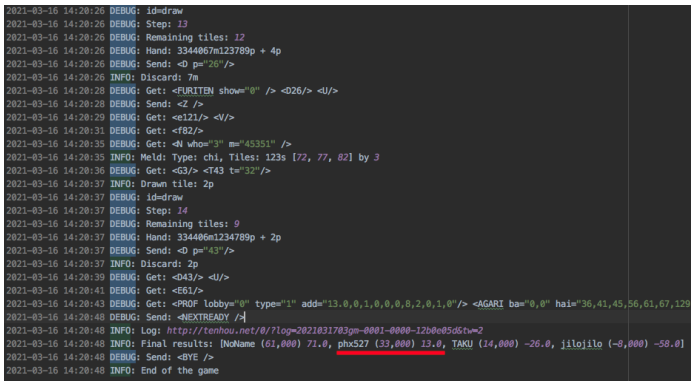


Fig. 10: Example run aginst three human players

the reinforcement learning phase, we deploy multiple agents with a local game manager to do the self-play. With the help of distribute framework, the number of worker can scale horizontally. We have the infrastructure for all three type of setup. During the game, our agent collect features from the known information in the table as the game progresses and transforms them into the single feature row in the format of the training dataset. This feature is used by models to make predictions. In discard model, the prediction will give us the distributions of probabilities of each class. Argmax applied to find out which one has the highest probability. The results are mapped to the corresponding unique tiles. We send a web-socket message from tenhou client to server and update the states accordingly. The other models have the similar mechanism , the difference is that we need to first manual check the condition for valid action. In other words, some rules need to be checked before we ask the model to make an action. Models only make predictions when they meet all the requirements. Fig.10 shown as an example run[3] against three human players. The red underline phx527 is our agent bot. Our current supervised models achieved 2nd place in this game. Agent plays relatively defensive against human players. It doesn't throw out any existing meld in hands and cautiously discard the tile that can't be used against self.

### D. Ranking

We evaluate our model on tenhou platform with about 50 games against other human player, our model is ranking in the 4 kyu. This ranking is purely for the supervised model. More games are needed in order to stabilize the ranking.

## VI. Conclusion

There are still some room for improvement for our model especially the reinforcement learning part. Due to the resources and time constraint, we are unable to finished the reinforcement learning training on time. We already setup our infrastructure of the distributed training using Ray. The performance of model should be more promising with the proximal policy optimization. This project is heavily focused on the engineering side, we spend most of time to setup the infrastructures of our supervised pipeline and distributed reinforcement learning framework. All the source code are available publicly for the researcher interested in this field to bootstrap the process of AI mahjong playing.


## Acknowledgment

We would like to express our sincere gratitude to our professor Dr. Michael Zyda for his valuable advice and guidance on our project. We also want to thank teach assistants team for their hard working on making everything happen.


[3] Example replay log

## References


[1] Ege Eren and A. Nihat Berker. Metastable reverse-phase droplets within ordered phases: Renormalization-group calculation of field and temperature dependence of limiting size. *Physical Review E*, 101(4), Apr 2020.

[2] Shiqi Gao, Fuminori Okuya, Yoshihiro Kawahara, and Yoshimasa Tsuruoka. Building a computer mahjong player via deep convolutional neural networks, 2019.

[3] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[4] Moyuru Kurita and Kunihito Hoki. Method for constructing artificial intelligence player with abstraction to markov decision processes in multiplayer game of mahjong, 2019.

[5] 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)*, pages 275–283, 2015.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.