

AI in Human-computer Gaming: Techniques, Challenges and Opportunities

Qi-Yue Yin^{1,2} Jun Yang³ Kai-Qi Huang^{1,2,4} Mei-Jing Zhao¹ Wan-Cheng Ni^{1,2}
Bin Liang³ Yan Huang^{1,2} Shu Wu^{1,2} Liang Wang^{1,2,4}

¹Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China

²School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China

³Department of Automation, Tsinghua University, Beijing 100084, China

⁴Center for Excellence in Brain Science and Intelligence Technology, Chinese Academy of Sciences, Beijing 100190, China

Abstract: With the breakthrough of AlphaGo, human-computer gaming AI has ushered in a big explosion, attracting more and more researchers all over the world. As a recognized standard for testing artificial intelligence, various human-computer gaming AI systems (AIs) have been developed, such as Libratus, OpenAI Five, and AlphaStar, which beat professional human players. The rapid development of human-computer gaming AIs indicates a big step for decision-making intelligence, and it seems that current techniques can handle very complex human-computer games. So, one natural question arises: What are the possible challenges of current techniques in human-computer gaming and what are the future trends? To answer the above question, in this paper, we survey recent successful game AIs, covering board game AIs, card game AIs, first-person shooting game AIs, and real-time strategy game AIs. Through this survey, we 1) compare the main difficulties among different kinds of games and the corresponding techniques utilized for achieving professional human-level AIs; 2) summarize the mainstream frameworks and techniques that can be properly relied on for developing AIs for complex human-computer games; 3) raise the challenges or drawbacks of current techniques in the successful AIs; and 4) try to point out future trends in human-computer gaming AIs. Finally, we hope that this brief review can provide an introduction for beginners and inspire insight for researchers in the field of AI in human-computer gaming.

Keywords: Human-computer gaming, AI, intelligent decision making, deep reinforcement learning, self-play.

Citation: Q. Y. Yin, J. Yang, K. Q. Huang, M. J. Zhao, W. C. Ni, B. Liang, Y. Huang, S. Wu, L. Wang. AI in human-computer gaming: Techniques, challenges and opportunities. *Machine Intelligence Research*, vol.20, no.3, pp.299–317, 2023. <http://doi.org/10.1007/s11633-022-1384-6>

1 Introduction

Human-computer gaming has a long history and has been a main tool for verifying key artificial intelligence technologies^[1, 2]. The Turing test^[3], proposed in 1950, was the first human-computer game to judge whether a machine has human intelligence. This has inspired researchers to develop AI systems (AIs) that can challenge professional human players. A typical example is a draughts AI called Chinook, which was developed in 1989 to defeat the world champion, and such a target is achieved by beating Marion Tinsley in 1994^[4]. Afterward, Deep Blue from IBM beat the chess grandmaster Garry Kasparov in 1997, setting a new era in the history of human-computer gaming^[5].

In recent years, we have witnessed the rapid develop-

ment of human-computer gaming AIs, from the DQN agent^[6], AlphaGo^[7], Libratus^[8], and OpenAI Five^[9] to AlphaStar^[10]. These AIs can defeat professional human players in certain games with a combination of modern techniques, indicating a big step in the decision-making intelligence^[11–13]. For example, AlphaGo Zero^[14], which uses Monte Carlo tree search, self-play, and deep learning, defeats dozens of professional go players, representing powerful techniques for large state perfect information games. OpenAI Five^[9], using self-play, deep reinforcement learning, and continual transfer via surgery, became the first AI to beat the world champions at an eSports game, displaying useful techniques for complex imperfect information games.

After the success of AlphaStar and OpenAI Five, which reach the professional human player level in the games StarCraft and Dota2, respectively, it seems that current techniques can solve very complex games. Especially the breakthrough of the most recent human-computer gaming AIs for games such as the Honor of Kings^[15] and Mahjong^[16] obeys similar frameworks of AlphaStar and OpenAI Five, indicating a certain degree of univer-

Review

Manuscript received on August 18, 2022; accepted on October 19, 2022; published online on January 7, 2023

Recommended by Associate Editor Mao-Guo Gong

Colored figures are available in the online version at <https://link.springer.com/journal/11633>

© The Author(s) 2023

salinity of current techniques. So, one natural question arises: What are the possible challenges of current techniques in human-computer gaming, and what are the future trends? This paper aims to review recent successful human-computer gaming AIs and tries to answer the question through a thorough analysis of current techniques.

Based on the current breakthrough of human-computer gaming AIs (most published in journals such as *Science* and *Nature*), we survey four typical types of games, i.e., board games with Go; card games such as heads-up no-limit Texas hold'em (HUNL), DouDiZhu, and Mahjong; first person shooting games (FPS) with Quake III Arena in capture the flag (CTF); real-time strategy games (RTS) with StarCraft, Dota2, and Honor of Kings. The corresponding AIs cover AlphaGo^[7], AlphaGo Zero^[14], AlphaZero^[17], Libratus^[8], DeepStack^[18], DouZero^[19], Suphx^[16], FTW^[20], AlphaStar^[10], OpenAI Five^[9], JueWu^[15], and Commander^[21]. A brief summary is displayed in Fig. 1.

The remainder of the paper is organized as follows. In Section 2, we describe the games and AIs covered in this paper. Sections 3–6 elaborate on the AIs for board games, card games, FPS games, and RTS games, respectively. In Section 7, we summarize and compare the different techniques utilized. In Section 8, we show the challenges in current game AIs, which may be the future research direction of this field. Finally, we conclude the paper in Section 9.

2 Typical games and AIs

Based on the recent progress of human-computer gaming AIs, this paper reviews four types of games and their corresponding AIs, i.e., board games, card games, FPS games, and RTS games. To measure how hard a game is to develop professional human-level AI, we extract several key factors that challenge intelligent decision-making^[22], which are shown in Table 1.

Imperfect information. Except for the board games, almost all the card games, FPS games, and RTS games are imperfect information games, which means that players do not know exactly how they come to the current states, e.g., current face in HUNL. Accordingly, players need to make decisions under partial observation. This leads to more than one node in an information set if the game is expanded into a tree. For example, the average information sets for the card games HUNL and Mahjong are 10^3 and 10^{15} , respectively. Moreover, compared with perfect information games such as Go, a subgame in an imperfect information game cannot be solved isolated from each other^[23], which makes solving the Nash equilibrium of imperfect information games more difficult^[24].

Long time horizon. In real-time games, such as StarCraft, Dota2, and Honor of Kings, a game lasts several minutes and even more than an hour. Accordingly, an

AI needs to make thousands of decisions. For example, Dota 2 games run at 30 frames per second for about 45 minutes, resulting in approximately 20 000 steps in a game if making a decision every four frames. In contrast, players in card games usually make fewer decisions. The long time horizon leads to an exponential increase in the number of decision points, which brings in a series of problems, such as exploration and exploitation, when optimizing a strategy.

In-transitive game. If the performance of different players is transitive, a game is called a transitive game^[25]. Mathematically, if v_t can beat v_{t-1} and v_{t+1} can beat v_t , v_{t+1} outperforms v_{t-1} . Then, a game is strictly transitive. However, most games in the real world are in-transitive. For example, in a simple game, “Rock-Paper-Scissor”, the strategy is in-transitive or cyclic. Commonly, most games consist of transitive and in-transitive parts, i.e., obey the spinning tops structure^[26]. The in-transitive characteristic makes the standardized self-play technique, widely used for agent ability evolution, fail to iteratively approach the Nash equilibrium strategy.

Multi-agent cooperation. Most board games and card games are purely competitive, where no cooperation between players is required. An exception is DouDiZhu, which needs two Peasant players playing as a team to fight against the Landlord player. In contrast, almost all real-time games, i.e., FPS games and RTS games, rely on players' cooperation to win the game. For example, five players in Dota2 and Honor of Kings form a camp to fight against another camp. Even though StarCraft is a two-payer competitive game, each player needs to control a large number of units, which need to cooperate well to win. Overall, how to obtain the Nash equilibrium strategy or a better-learned strategy under multi-agent cooperation is a hard problem because specially designed agent interaction or alignment needs to be considered.

In summary, different games share different characteristics and aim to find different kinds of solutions, so distinct learning strategies are developed to build AI systems. In Sections 3–6, we will see that behind the game types is the evolution of techniques that are designed for perfect information, imperfect information, and more complex real-time and long-time horizon imperfect information games. So, a taxonomy based on different kinds of games is utilized. Finally, in this paper, the AIs cover: AlphaGo, AlphaGo Zero, and AlphaZero for the board game Go; Libratus, DeepStack, DouZero, and Suphx for card games HUNL, DouDiZhu, and Mahjong, respectively; FTW for the FPS game Quake III Arena in capture the flag model; AlphaStar, Commander, OpenAI Five, and JueWu for StarCraft, Dota2 and Honor of Kings, respectively.

3 Board game AIs

The AlphaGo series is built based on Monte Carlo

¹ A name known by the public.

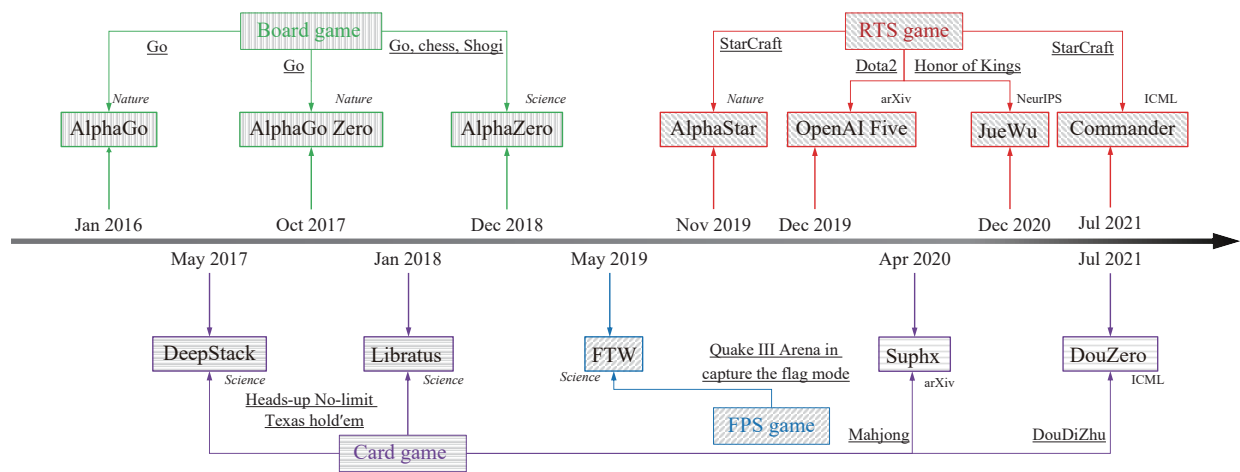


Fig. 1 Games and AIs surveyed in this paper

Table 1 Characteristics of four typical kinds of games

Games	Board games		Card games		FPS games		RTS games	
	Go series	HUNL	DouDiZhu	Mahjong	CTF	StarCraft	Dota2	Honor of Kings
Imperfect information	×	✓	✓	✓	✓	✓	✓	✓
Long time horizon	✓	×	×	×	✓	✓	✓	✓
In-transitive game	✓	✓	✓	✓	✓	✓	✓	✓
Multi-agent cooperation	×	×	✓	×	✓	✓	✓	✓

tree search (MCTS)^[27, 28], which is widely utilized in previous Go programs. AlphaGo came out in 2015 and beats European Go champion Fan Hui by 5 : 0, which was the first time that an AI won against professional players in a full-size game, Go without Renzi. Afterward, an advanced version called AlphaGo Zero was developed using different learning frameworks, which needs no prior professional human confrontation data and reaches superhuman performance. AlphaZero uses a similar learning framework to AlphaGo Zero and explores a general reinforcement learning algorithm, which masters Go along with another two board games, chess, and Shogi. A brief summarization is shown in Fig. 2.

3.1 MCTS for AlphaGo series

One of the key factors of the AlphaGo series is MCTS, which is a typical tree search-based method. Generally, a simulation of MCTS consists of four steps, repeated hundreds and thousands of times for one step decision. The four steps consist of selection, expansion, evaluation, and backup, which are operated in a tree as shown in the lower right corner of Fig. 2. In the selection step, a leaf node is set starting from the root node, i.e., the state where an action needs to be decided, based on the evaluation of the nodes in the tree. Next is the expansion of the tree by adding a new node. Finally, starting from the expanded node, a rollout is performed to obtain a value

for the node, which is used to update the values of all nodes in the tree.

In the AlphaGo series, traditional MCTS is improved via deep learning to limit the width and depth of the search so as to handle the huge game tree complexity. Firstly, in the selection stage, a node is selected based on the sum of the action value Q and a bonus $u(p)$. The action value is the average node value of all simulations, and the node value is the evaluation of a node based on the predication of the value network and the rollout results based on the rollout network. The bonus is proportional to the policy value (probability of selecting points in Go) calculated via the policy network, but inversely proportional to the visit count. Secondly, in the expansion stage, a node is expanded and its value is initialized through the policy value. Finally, when making an estimate of the expanded node, the rollout results based on the rollout network and the predicted results based on the value network are combined. As noted in AlphaGo Zero and AlphaZero, the rollout is removed and the evaluation of the expanded node is based solely on the prediction results of the value network, which will be explained in the following subsection.

3.2 Learning for AlphaGo series

3.2.1 Learning for AlphaGo

Learning of AlphaGo consists of several steps. Firstly,

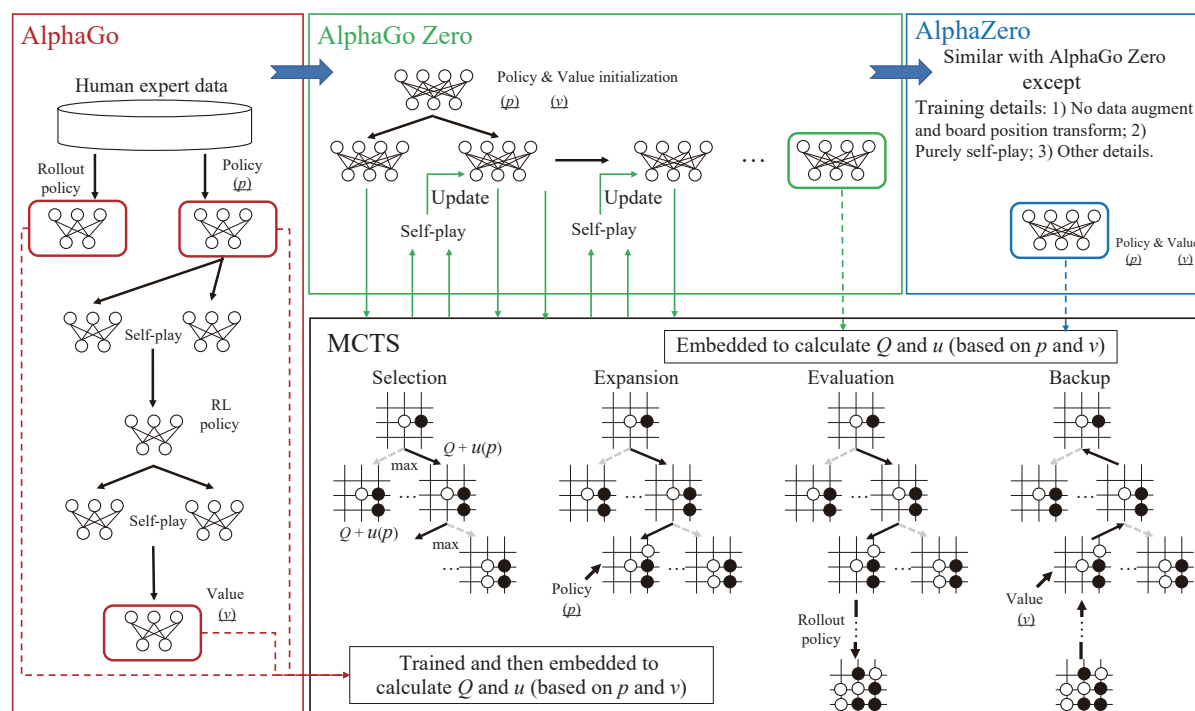


Fig. 2 A brief framework of the AlphaGo series

a supervised learning policy network and a rollout policy network are trained with human expert data, which outputs the probability of the next move position based on 160 000 games played by KGS 6 to 9 dan human players. The differences between them are the neural network architectures and features used. Specifically, the supervised policy consists of several convolutional layers using a $19 \times 19 \times 48$ image stack of 48 feature planes as input, whereas the rollout policy is just a linear softmax policy using some less fast, incrementally computed, local pattern-based features. With the above high-quality data, a very good initiation of the supervised learning policy network is obtained, which reaches amateur level, i.e., about amateur 3 dan (d).

With the supervised learning policy network trained, a reinforcement learning policy network is initialized (with the same network) and then improved through self-play, which uses the network of the current version to fight against its previous versions. Based on conventional policy gradient methods to maximize the winning signal, the reinforcement learning policy network reaches better performance than the supervised learning network, i.e., an 80% winning rate against the supervised learning policy.

In the third step of AlphaGo, a value network is trained to evaluate the state, which shares the same features and neural network architecture with the supervised learning policy network except for the last two layers due to different output dimensionalities. Especially, a dataset consisting of 30 million state-outcome pairs is collocated through the self-play of the reinforcement learning network. Then, a regression task is developed by min-

imizing the mean squared error between the predicted result of the value network and the corresponding outcome (win or loss signal). With the value network, MCTS can reach a better performance than just using the supervised learning policy network. Finally, the well-trained supervised learning policy, value network, and rollout network are embedded into MCTS, which reaches a professional level of 1 to 3 dan (p).

3.2.2 Learning for AlphaGo Zero and AlphaZero

Unlike AlphaGo, whose policy network and value network are trained through supervised learning and self-play between the policy networks, AlphaGo Zero trains policy and value networks through self-play of MCST embedded in the current version of the networks. Besides, different neural network architectures are adopted compared with AlphaGo, i.e., residual networks. As for the input, more simplified features are used without considering the human player experience. AlphaZero shares the same learning framework as AlphaGo Zero. Overall, they consist of two alternating repetition steps: automatically generating data, policy and value networks training.

When generating training data, self-play of MCTS is performed. MCTS embedded in the current policy and value networks is used to select each move for the two players at each state. Generally, MCTS selects an action based on the maximum count, but AlphaGo Zero makes it a probability to explore more actions by normalizing the count. Accordingly, state-move probability pairs are stored. Finally, when a game ends, the winning signal (+1 or -1) is recorded for value network training.

Relying on the collected state-move probability and winning signal, the policy and value networks are trained.

More specifically, the distance between the predicted probability of the policy network and the collected probability for each state is minimized. Besides, the distance between the predicted value of the value network and the winning signal is minimized. The overall optimization objective also contains an $L2$ weight regularization to prevent overfitting.

3.2.3 Learning differences

Based on MCTS, deep learning, reinforcement learning, and self-play are nicely evolved in the AlphaGo series, as shown in Fig. 2. The main difference is the learning frameworks utilized, elaborated in the following paragraphs. To sum up, AlphaGo uses human expert data to obtain the supervised policy network, based on which self-play between supervised policy network is performed to obtain reinforcement learning policy and the subsequent value network based on similar self-play of reinforcement learning policy, and all the trained networks are embedded into MCTS for decision making. However, AlphaGo Zero uses no human expert data and trains the policy and value networks based on data generated through self-play of MCTS embedded in the current version of policy and value networks. AlphaZero shares the same training framework as AlphaGo Zero, except for several small training settings.

Apart from the training framework, there are several factors in which AlphaGo Zero differs from AlphaGo. Firstly, no rollout policy network is used to evaluate the expanded node, and the benefit is a speedup of the MCTS simulation. With the higher quality data generated by the new learning framework, values of leaf nodes can be better estimated without using a rollout policy. Besides, no human expert data are utilized for deep neural network training. Secondly, the policy and value networks in AlphaGo Zero share most of the parameters (convolutional layers) instead of two separate networks, which shows a better Elo rating^[29]. What's more, residual blocks, as a powerful modular for deep learning, is utilized in AlphaGo Zero, and shows much better performance than just using convolutional blocks as in AlphaGo. Finally, the input to the policy of AlphaGo Zero is a $19 \times 19 \times 17$ image stack instead of the $19 \times 19 \times 48$ image stack, which rarely uses human engineering features compared with AlphaGo, e.g., the designed ladder capture and ladder escape features.

AlphaZero aims to develop a more general reinforcement learning algorithm for various board games such as Go, chess, and Shogi. Since the rules of chess and Shogi are very different from Go, AlphaZero makes several changes to the training details to fit the above goal. As for the game Go, there are two main training details that are different from AlphaGo Zero. Firstly, no data augment and transformations such as rotation or reflection of the positions are applied. Secondly, AlphaZero uses a pure self-training framework by maintaining only a single neural network instead of saving a better model in each

iteration of training.

4 Card game AIs

The Card game, as a typical in-perfect information game, has been a long-standing challenge for artificial intelligence. DeepStack and Libratus are two typical AI systems that defeat professional poker players in HUNL. They share the same basic technique, i.e., counterfactual regret minimization (CFR)^[30]. Afterward, researchers are focusing on Mahjong and DouDiZhu, which raise new challenges for artificial intelligence. Suphx, developed by Microsoft Research Asia, is the first AI system that outperforms most top human players in Mahjong. DouZero, designed for DouDiZhu, is an AI system that was ranked first on the Botzone leaderboard among 344 AI agents. A brief introduction is shown in Fig. 3.

4.1 DeepStack and Libratus for HUNL

HUNL is one of the most popular poker games in the world, and plenty of world-level competitions are held every year, such as the World Series of Poker. Before DeepStack and Libratus came out, HUNL was a primary benchmark and challenge of imperfect information games with no AIs that had defeated professional players.

4.1.1 CFR for DeepStack and Libratus

Since being proposed in 2007, CFR has been introduced in poker games. CFR minimizes counterfactual regret for large extensive games, which can be used to compute a Nash equilibrium. Generally, it decomposes the regret of an extensive game into a set of additive regret terms on information sets that can be minimized independently. Due to the high cost of time and space, basic CFR is not applicable to HUNL, which is much more complex than limited poker. Various improved CFR approaches have been developed, considering improving computing speed or compressing the required storage space^[31, 32]. For example, based on CFR, continue-resolving^[18], and safe and nested subgame solving^[8], are key factors for success of the DeepStack and Libratus, respectively.

4.1.2 Learning for DeepStack

The key to learning for DeepStack is continual re-solving, which is assisted by depth-limited look-ahead via deep learning and sparse look-ahead trees. Re-solving, begins with a strategy and reconstructs the strategy by resolving it every time a decision is required. To accomplish this at any decision point, DeepStack maintains a player's own range and opponent counterfactual values. Giving three specific updating rules on own action, chance action, and opponent action ensures that opponent counterfactual values are properly bounded. A very important characteristic is that there is no requirement for knowledge of opponent action and range to update the above values, which makes DeepStack very efficient.

However, purely re-resolving is intractable because of

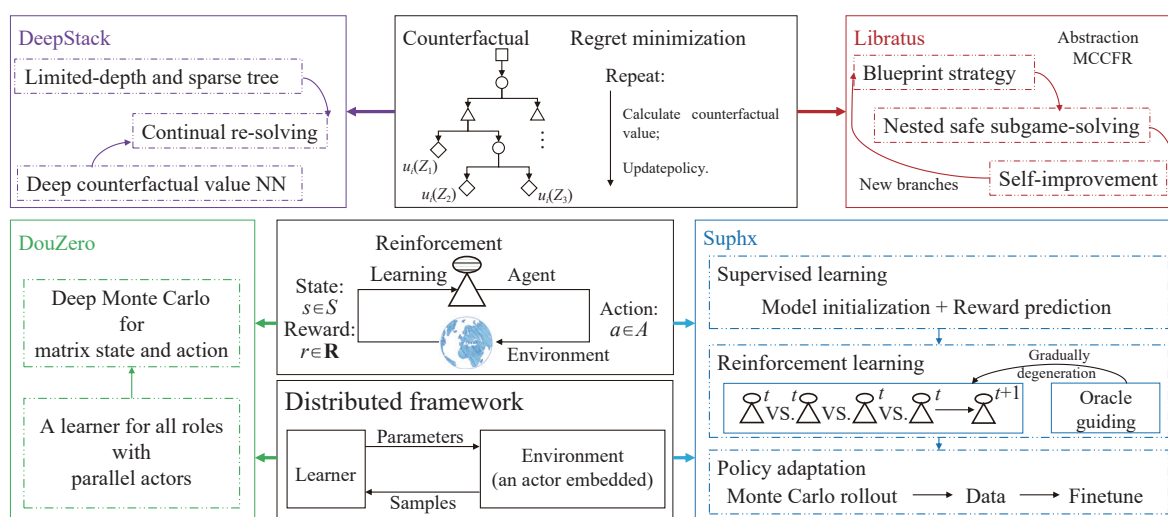


Fig. 3 A brief framework of Card game AIs

the deep depth of the game tree in HUNL. To handle this problem, Deepstack restricts the depth of the subtree via intuition. A counterfactual value function is trained with a deep neural network that uses a standard feedforward network with seven fully connected hidden layers, which is utilized for estimating how valuable holding certain cards is. Moreover, by limiting actions to fold, call, two or three-bet actions, and all-in, the resolved games are reduced to have about 10^7 decision points, largely reduced compared to 10^{160} decision points for the whole game. Based on such an abstraction, DeepStack can make a decision in no more than 5 seconds on a machine with a single NVIDIA GeForce GTX 1 080 graphics card.

4.1.3 Learning for Libratus

The learning of Libratus does not require expert domain knowledge and consists of three main steps: building a blueprint strategy, nested safe subgame solving, and self-improvement. Blueprint strategy is solved by an improved version of CRF, i.e., Monte Carlo CFR, for an abstracted game, which provides a strategy for the early rounds of the game and an approximation for the latter rounds. As for the abstraction, certain bet sizes are abstracted based on an application-independent parameter-optimization algorithm. However, no card abstraction on the first and second betting rounds is adopted, where the decision strategy is purely based on the blueprint strategy.

Nested safe subgame solving is used in the third and fourth betting rounds, which provides a real-time solution for a more detailed abstraction of the game tree. The abstraction in the blueprint is relaxed instead of rounding the bet size to the nearest one. Libratus will make a distinct strategy in response to off-tree actions. Nested safe subgame solving ensures that new strategy for the subgame improves the blueprint strategy by making the opponent worse off no matter what cards the opponent is holding. Finally, self-improvement computes a game-theoretic strategy for branches that are added based on the

actual moves of opponents.

4.1.4 Learning differences

Intuitively, DeepStack solves the subtree based on re-solving assisted by deep neural networks for counterfactual values prediction, whereas, Libratus utilizes a nested safe subgame solving strategy to improve the original abstraction-based strategy. Both methods use estimated value instead of the upper bound value of the opponent, but Libratus claims that DeepStack does not share its improvement of de-emphasizing hands.

Libratus plays the first two rounds based on precomputed blueprint strategy, which makes a big abstraction of opponent actions. However, DeepStack re-solves each subgame no matter what rounds it is now deciding, making it more flexible in dealing with opponent off-tree actions. To make Libratus more powerful in handling off-tree opponent bet sizes in the first two rounds, a self-play improvement modular is designed based on the actual moves of the opponent, which can largely remedy defects.

4.2 Suphx and DouZero for Mahjong and DouDiZhu

Unlike HUNL, Mahjong has different types of action, and the regular order of plays can be interrupted, making the game tree consist of a huge number of paths between the consecutive actions of a player. This leads to the successful MCTS and CFR-based techniques for Go and HUNL not being the best choice. Similarly, the actions of DouDiZhu are complex and can not be abstracted, making the above methods hard to be applied. To this end, Suphx and DouZero adopt deep reinforcement learning as a basic tool for AI development, which aims to reach high-level performance and cares little about the characteristics of the solution, such as the Nash equilibrium.

4.2.1 Basic techniques for Suphx and DouZero

Reinforcement learning (RL) is a typical type of ma-

chine learning^[33], which has become one of the most important decision-making techniques since the breakthrough of AlphaGo^[34]. Generally, RL follows the framework of policy evaluation and policy improvement by interacting with the environment. Because of the trial and error mechanism, RL requires a huge amount of data for policy learning, leading to sample inefficient problems^[35, 36]. Distributed training^[37], which utilizes multiple machines for learning a task, is now combined with RL to alleviate the above problem^[38, 39].

Nair et al.^[40] proposed the first massively distributed architecture for RL, which consists of four components. The first part is parallel actors, which are used to interact with multiple environments and generate data; the second component is parallel learners that consume data for policy training; the third and fourth parts are distributed neural networks and store of experience to connect the actor and learner. Based on the above framework, a number of advanced distributed reinforcement learning frameworks are developed, and data throughput is largely improved^[41–43]. In Suphx and DouZero, distributed learning is adopted to accelerate RL training, where multiple rollouts are concurrently performed to collect data.

4.2.2 Learning for Suphx

Suphx is a hybrid learning system, which consists of a rule-based winning model and five learning-based networks (used for basic actions to discard, Riichi, Chow, Pong, and Kong in Mahjong) to form the decision flow. The five networks use almost the same convolutional blocks, except for the dimensions of the input and output layers, due to their characteristics. Generally, training of the five learning-based networks contains three major steps: supervised learning, self-play reinforcement learning, and a run-time policy adaptation.

Supervised learning is performed utilizing state-action pairs collected from human players in Tenhou platform², and then acts as initialization for the self-play reinforcement learning stage. Usually, each game consists of multiple rounds, and the final reward signal is obtained by accumulating all the round scores, so it is hard to guide reinforcement learning in each round because some players may tactically lose several rounds to win the game. In Suphx, this problem is solved by using a GRU network to predict the feedback for each round. More specifically, data from top human players are collected as reward and a regression-based objective is constructed between past and present round information and the final game reward. When performing reinforcement learning, this prediction is served as the intermediate reward for each round in a game.

In the reinforcement learning stage, considering that learning is slow, facing the rich hidden information in Mahjong, Suphx proposes a method called oracle guiding. Firstly, an oracle agent is trained by using the perfect information, including unobservable private tiles of all the

players and the tiles on the wall. Since a simple knowledge distillation method does not work because it is hard for a normal agent with very limited information to mimic the oracle agent, Suphx gradually drops out the perfect features so that the oracle agent can slowly degenerate into the normal agent.

Run-time policy adaptation is utilized so that the learned policy can be properly adapted based on the tiles of the current round. The motivation comes from the human player, who will act very differently based on different tiles at the beginning of each round. A parametric Monte Carlo policy adaption approach is proposed, which consists of two steps. Firstly, Suphx simulates multiple games by self-play using a previously trained policy at the beginning of a round, with which trajectories are collected. Then, gradient updates are performed using the about data for policy finetune. Based on the experimental results, the simulation numbers do not need to be very large, and in every round, the policy adaption can be adopted.

4.2.3 Learning for DouZero

In DouZero, a deep Monte Carlo method is developed with specially designed matrix-form state and action spaces. Since there are up to 27 472 possible actions for a player, a matrix-form action representation provides a nice way to encode, and more importantly, reason about unseen actions. This is one of the key factors that DouZero can handle huge action space. Considering Monte Carlo approaches are usually inefficient because of their high variance issue, DouZero utilizes distributed training to parallelize the data generation part. Especially, a lot of actors are raised, with each maintaining the local networks of the three players and generating episode trajectories, based on which a learner of global networks for the three players is trained. As for the Q-network used in the Monte Carlo approach, an LSTM and six layers of multi-layer perception are utilized to map historical moves (matrix-form) and action (matrix-form) to the Q value. Overall, the training algorithm of DouZero is simple and efficient, and the authors show that classic Monte Carlo methods can be properly designed to deal with games with a complex action space.

4.2.4 Learning differences

In addition to utilizing reinforcement learning algorithms and distributed frameworks for training acceleration, the learning frameworks of DouZero and Suphx are very different. Firstly, the training of Suphx is a complex and multi-stage system, whereas the training of DouZero is relatively simple with a distributed deep Monte Carlo method. In Suphx, data from top human players are required for network initialization and round reward prediction, which is important for the whole AI. However, in DouZero, no human data is required, and networks for different players are trained from scratch, based on which it ranks the first in the Botzone leaderboard among 344 AI programs.

² <https://tenhou.net/man/>

5 First-person shooting game AIs

CTF is a typical three-dimensional multiplayer first-person video game in which two opposing teams are fighting against each other on indoor or outdoor maps. As we will see in the next section, the settings for CTF are very different from the current multiplayer video games. More specifically, agents in CTF cannot access the state of other players, and agents in a team cannot communicate with each other, making this environment a very good testbed for learning agents to emerge communication and adapt to zero-shot generation. Zero-shot means that an agent cooperated or confronted is not the agent trained, which can be human players and arbitrary AI agents. Based solely on pixels and game points like a human as input, the learned agent FTW reaches a strong human-level performance. A brief introduction is shown in Fig. 4.

5.1 Learning framework

The aim of FTW is to train agents that can adapt to the variability of maps, the number of agents, and the choice of teammates and opponents. To achieve such high scalability, conventional self-play methods are claimed to be unstable, and those approaches in their basic form cannot support concurrent training, which is important for scalability. To handle the problems, FTW trains in parallel a population of agents, where each agent is trained based on distributed reinforcement learning with experiences collected by dynamically selected teammates and opponents. Moreover, an online evolutionary algorithm is developed guiding agents learning so as to direct the population. The above processes are called population-based training, which will be elaborated on in the following subsection.

Considering that the global reward is sparse for FTW, which provides a signal after lasting for 4 500 frames. FTW learns several intermediate rewards to accelerate training. A key problem in learning such rewards is to ensure that the optimization of intermediate rewards pro-

motes the policy optimization for chasing global rewards. This problem is solved by a specially designed joint maximization objective^[44], where inner optimization optimizes intermediate rewards through distributed reinforcement learning, and outer optimization, regarded as a meta-game, is optimized through population-based training for transformation between intermediate reward and global reward.

Another specific aspect of FTW lies in its neural network design. Due to the partial observation of the agent, FTW follows the idea of reinforcement learning as probabilistic inference. Accordingly, a hierarchical LSTM network with different timescales is developed, where the LSTM with a fast timescale generates hidden states and is enhanced by the LSTM with a slow time scale. The hidden states of the LSTM with the fast timescale are then severed as the variational posterior for the final action selection.

5.2 Population-based training

Population-based training maintains a population of agents, which consists of two important components to direct the learning process: sample teammates and opponents for an agent to generate data, reset and perturb hyperparameters, and transformation parameters for underperforming agents based on the training process.

When collecting training data for agent policy optimization, a sampling method based on Elo^[29] of agents is utilized. It encourages agents with similar skills (Elo scores) to be teammates and opponents, ensuring that the outcome of a game is sufficiently uncertain so as to guide agent learning useful policy. Since the conventional Elo calculation method is designed without considering agent cooperation, FTW makes the assumption that the rating of a team can be decomposed as a sum of skills for a team. With the above assumption, the Elo for each agent can be obtained following the regular Elo optimization approach.

After training a generation of population, hyperpara-

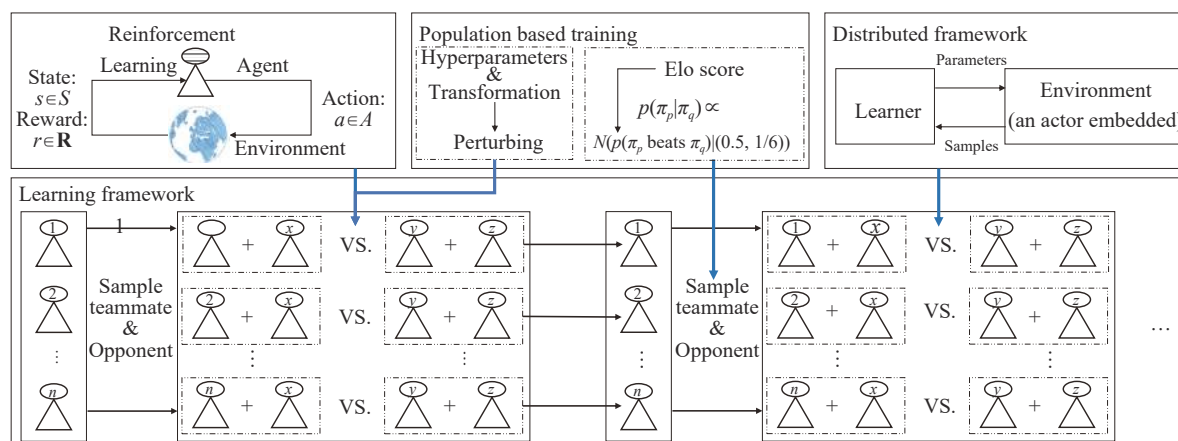


Fig. 4 A brief framework of FTW for the game CTF

meters like learning rate and transformation parameters between the intermediate rewards and global rewards for underperforming agents are reset and perturbed by using the better-performing agents as a reference. More specifically, if an agent with a team cannot win another agent with a team (e.g., 70% winning rate), the losing agent copies the policy, reward transformation, and hyperparameters of the better agent, and then probabilistically perturbs the inherited values with a small range, e.g., $\pm 20\%$ with a probability of 5%. The above exploration process helps to find better hyperparameters and transformation parameters.

6 RTS game AIs

The RTS game, as a typical kind of video game, with tens of thousands of people fighting against each other, naturally becomes a testbed for human-computer gaming. Furthermore, RTS games usually have a complex environment, which captures more of the nature of the real world than previous games, making the breakthrough of such games more applicable. AlphaStar, developed by DeepMind, uses general learning algorithms and reaches the grandmaster level for all three races for StarCraft, which also outperforms 99.8% of human players who are active on the European server (about 90 000 players). Commander, as a lightweight computation version, follows the same learning architecture as AlphaStar, which uses an order of magnitude less computation and beats two grandmaster players in a live event. OpenAI Five aims to solve the Dota2 game, which is the first AI system to defeat the world champions in an eSports game. As a relatively similar eSports game to Dota2, Honor of Kings shares most similar challenges, and JueWu becomes the first AI system that can play full RTS games

instead of restricting the hero pool. A brief introduction is shown in Fig. 5.

6.1 Basic techniques for RTS game AIs

To handle complex RTS games, reinforcement learning accelerated by distributed framework becomes a basic tool. Different from the distributed frameworks designed for Suphx and DouZero, a larger data throughput framework is designed because a huge interaction with the environment is required. Previous distributed reinforcement learning mainly maintains two important modular: parallel environments, with each embedded an actor to generate actions, and learners to consume data collected by the environments for policy updating. With such a distributed framework, plenty of time is wasted, because in each environment, a model inference should be performed for a single action. Current distributed reinforcement learning performs centralized model inference for states collected from multiple environments and distributes actions for each environment, as shown at the bottom of Fig. 5. Based on the learner-centralized actor-environment architecture, model inference time is largely reduced, which will save time for big models used for complex games.

6.2 Learning for AlphaStar

The learning of AlphaStar consists of two main steps: supervised learning to initialize agent parameters and multi-agent reinforcement learning to improve the agent. The architecture of the network used for the above two steps consists of scatter connections for spatial and non-spatial features, an LSTM to remember past states, and an auto-regressive and recurrent pointer network for the

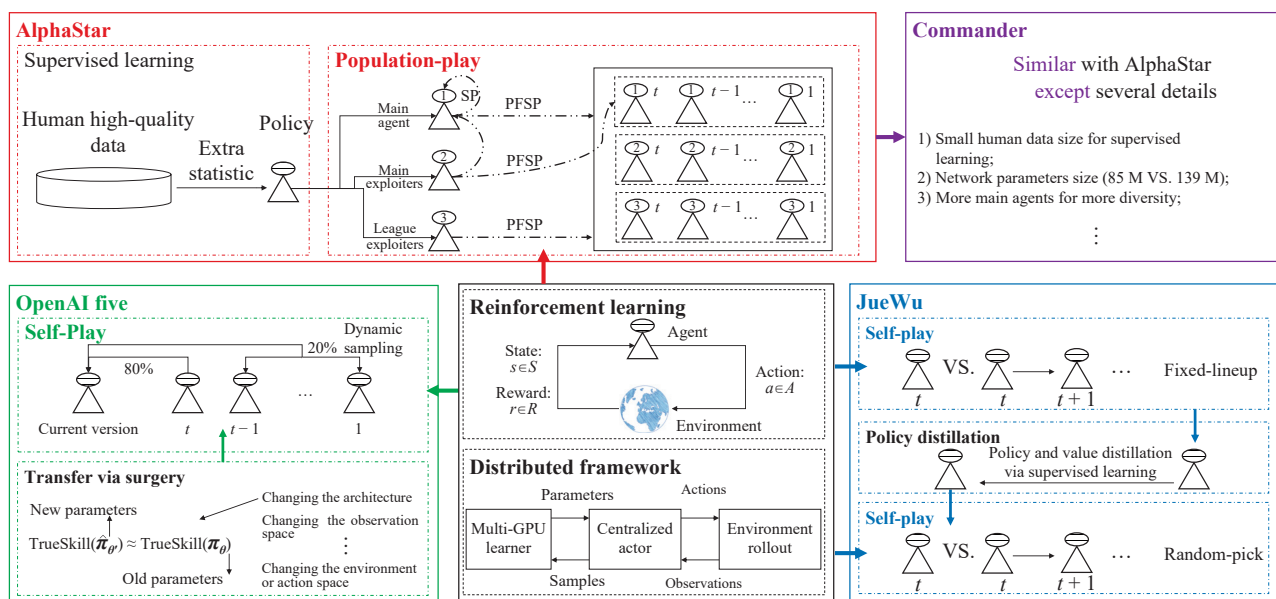


Fig. 5 A brief AI framework for typical RTS games

specifically designed combinatorial action. In supervised learning, a high-quality dataset³ is collected to train the agent parameters. The dataset consists of 971 000 replays from human players whose MMR scores (Blizzard's metric) are greater than 3 500, i.e., in the top of 22% of players. Since there are three races for StarCraft, AlphaStar trains one agent for each race. To fully explore the human experience, especially in the game beginning, where little combat feedback can be obtained, AlphaStar extracts a statistic variable to condition the policy and adopts Kullback-Leibler divergence between human actions and the policy outputs to assist learning. Such a statistic variable encodes each player's first 20 buildings and units, which reflects a type of opening strategy for AlphaStar. After the above supervised training, AlphaStar fine tunes the policy using a subset but more professional human player data (with MMR above 6 200), which improves the policy by 9% when fighting against the built-in elite bot.

After supervised learning for agent initialization, a multi-agent reinforcement learning framework with league training is developed so as to alleviate the game-theoretic challenges, such as cycles between strategies. We first introduce agent types in the league and then elaborate on how to train different agents. The league has three types of agents for each race: main agent, main exploiter, and league exploiter. The training of these agents lies in how to select opponents in the league and whether or not to reset the learned parameters. Specifically, opponents of the main agent are the main agent itself and all agents in the league so as to be strong enough for the final testing. Opponents of the main exploiter are the current main agent and the previous main agent versions to find weaknesses of the main agent. Opponents of the league exploiter are all agents in the league to discover possible weaknesses of the entire league. With the main exploiter and the league exploiter added to the league, the training of the main agent can properly overcome the weakness of itself and in the league. Usually, the evolution of the league can be called a kind of population-play, which is different from self-play that only maintains one agent.

When deciding sampling probabilities of opponents for different types of agents, an improved version of fictitious self-play called prioritized fictitious self-play is designed, which selects opponents based on winning rate against the agent instead of a uniform mixture of opponents. Detailed probability distribution and calculation can be found in the original paper. Note that when a generation of main exploiter or league exploiter agent is obtained, it is periodically reinitialized to supervised learned agent so as to extend the diversity of the league.

6.3 Learning for OpenAI Five

The learning of OpenAI Five is based on distributed

³ <https://github.com/Blizzard/s2client-proto>

self-play deep reinforcement learning. The architecture of the agent is mainly based on a 4 096-unit LSTM, which maps the processed observation to the action and value heads used for the proximal policy optimization algorithm. With their distributed learning system, OpenAI Five successfully extends the learning batch size to 2 949 120 time steps, which are important for training. When performing self-play to generate training data, the agent plays against itself for 80% of the games and against past versions for 20% of the games. Modifying conventional self-play in this way avoids strategy collapse and ensures that the learned agent is robust to a wide range of opponents. To effectively sample opponents from a large number of past versions, OpenAI Five maintains a score for each agent and changes the score based on the winning signal of the training trajectories. This strategy makes sure a dynamic sampling is performed to select useful agents to play against.

Another key factor for the success of OpenAI Five is a tool called continual transfer via surgery, which adjusts the parameters of a learned model for adapting to a new version of Dota2. Such a tool is essential because Valve usually publishes a new version of Dota2 every few months, resulting in performance degradation of the learned model. Even though a new model can be trained from scratch, the time is limited, and the resource consumption is intolerable. What's more, the designed tool makes the training of the agent more efficient because model parameters and architectures can be adjusted based on performance in the training process. Parameter transfer obeys a basic rule, i.e., the TrueSkill of the new agent (new parameter space) matches that of an already learned agent. Based on such principle, OpenAI Five develops different methods for changes in the architecture, observation space, action space, etc.

6.4 Learning for JueWu

The learning of JueWu is similar to that of OpenAI Five, where no human player data is utilized for agent initialization. However, to play with a hero pool of full RTS games instead of restricting the selection of heroes, JueWu developed a new training framework compared with the basic form of OpenAI Five. More specifically, learning of JueWu consists of three main steps: fixed-lineup training, multi-teacher policy distillation, and random-pick training, followed by an MCTS based approach for learning to draft. All the steps utilize a similar network architecture to that of AlphaStar, except that multi-head values are used for grounding the structural rewards that are related to the game.

Considering the self-play of massive disordered agent combinations makes training an agent a very hard task, JueWu adopts a curriculum-based training scheme: firstly using fixed-lineup and then utilizing random pick. Several fixed lineups without hero repeat are carefully selected,

based on which, distributed reinforcement learning is performed to train several teacher agents. To generate such lineups, JueWu analyzes vast amount of human player data, and selects relatively balanced teams. Based on the teacher agents, a policy distillation is conducted to learn a bigger student agent. The distillation is modeled as a supervised learning framework to minimize the difference between outputs of teacher and student models, i.e., Shannon's cross entropy between action distributions and Euclidean distance between value estimations. Finally, based on the student agent, another distributed reinforcement learning is applied for random pickups. Student agents, learned from fixed-lineup and served as initialization of random pick, largely reducing training difficulty.

A very important and interesting part of RTS games like Dota2 and Honor of Kings is hero drafting to form two teams. JueWu proposes an MCTS and neural network-based approach to handle the problem of a huge combination of agents, i.e., more than 10^{11} . The motivation for using a neural network in MCTS is similar to AlphaGo Zero, which is to estimate the value of the expanded node without using a time-consuming rollout. Unlike OpenAI Five, the terminal state of the draft is not the end of a game, so a winning or losing signal cannot be obtained. To construct a dataset for the training value estimation network, the label, i.e., winning signal, should be obtained. To solve this problem, JueWu collects another dataset, which performs plenty of matches using randomly selected teams with the learned reinforcement learning model. Then, a lineup-winning result dataset is developed, based on which a winning prediction network can be trained and used as a signal for value network training labels.

6.5 Learning for Commander

Commander adopts a very similar learning framework of AlphaStar for agent learning (including the network architecture), i.e., supervised learning followed by multi-agent reinforcement learning. The main differences are several important details, which make Commander beat two professional players with an order of magnitude less computation. Firstly, Commander uses a much smaller human player dataset, based on which learning rate, batch size, multi-stage training, and network structure are carefully designed for supervised learning. In multi-agent reinforcement learning, Commander devises the training loss and uses more main agents for more diversity, which improves the learning efficiency.

6.6 Learning difference

Nowadays, deep reinforcement learning accelerated by distributed learning has become a general method to train high-performance AIs. Apart from this, the four typical AIs, i.e., AlphaStar, OpenAI Five, JueWu, and Com-

mander, share several differences.

Firstly, to train each generation of agents, those AIs utilize self-play (or revised self-play) or population-play mechanisms. In JueWu and OpenAI Five, a relatively simple self-play is performed to train each generation of agents. To avoid strategy collapse and ensure that the learned agent is robust to a wide range of opponents, a certain percentage of past versions are usually selected as opponents. This selection can be specially designed instead of using fictitious self-play, i.e., uniformly selecting past versions. For example, OpenAI Five selects past versions with 20% of rollout games. AlphaStar utilizes a prioritized fictitious self-play mechanism to select opponents, based on which relatively hard agents and agents with similar levels are more likely to be chosen. What's more, AlphaStar and Commander adopt league training, which is a powerful population-play compared with self-play for more diverse agent learning.

Secondly, purely based on reinforcement learning usually requires huge computational resources because of its trial and error mechanism, so those AIs utilize human player data to assist reinforcement learning. In AlphaStar and Commander, supervised learning based on high-quality data is performed to initialize policy networks to provide a good and diverse initialization for reinforcement learning. What's more, statistics are extracted from human data to constrain the policy in the reinforcement learning stage, which helps a lot based on the ablation study in their papers. In JueWu, human data is not used for policy initialization. Instead, the data is used to analyze the hero lineups to provide relatively balanced teams for the first learning stage, i.e., self-play reinforcement learning with a fixed-lineup. In OpenAI Five, no human data is utilized, and OpenAI just uses self-play reinforcement learning for policy training, using huge computational resources for over 10-month training.

Thirdly, several new techniques are developed to overcome some challenging problems in the games. Different from population-based training in FTW, AlphaStar maintains a league for agent training, where different types of agents are responsible for different tasks. Even though being heuristic, league based multi-agent training provides a very useful idea for complex real-time games with game-theoretic challenges. Continual transfer via surgery is very useful as an effective tool to make full use of the currently learned model for changing environments, because the real world environment is inevitably changing over time. Such a technique can largely reduce computation costs, and change models when necessary.

7 Discussions

Based on the current breakthrough of human-computer gaming AIs, currently utilized techniques can be roughly divided into two categories, i.e., tree search (TS) with self-play (SP) and distributed deep reinforcement

learning (DDRL) with self-play or population-play (PP). It should be noted that we just mention the basic or key techniques in each category, based on which different AIs usually bring in other key modules based on the games, and those new modules are sometimes not generic across games. As shown in Table 2, the tree search has two kinds of representative algorithms: MCTS, usually used for perfect information games, and CFR, conventionally designed for imperfect information games. As for population-play, it is used for three situations: different players/agents do not share the same policy network due to the game characteristics (e.g., landlord and peasants agents in DouZero); populations can be maintained to overcome the game theoretical challenges such as non-transitivity (e.g., main agent, main exploiter, and league exploiter in AlphaStar); populations combined with population-based training to learn scalable agents (e.g., populations in FTW). With the comparison, we discuss two points as follows.

Table 2 Techniques comparison of different AIs

Category	Sub-category	AIs
TS+SP	MCTS+SP	AlphaGO ^[7] , AlphaGo Zero ^[14] , AlphaZero ^[17]
	CFR+SP	DeepStack ^[18] , Libratus ^[8]
DDRL+SP/PP	DDRL+SP	Suphx ^[16] , OpenAI Five ^[9] , JueWu ^[15]
	DDRL+PP	DouZero ^[19] , FTW ^[20] , AlphaStar ^[10] , Commander ^[21]

7.1 How to reach Nash equilibrium?

Nash equilibrium^[45], an important concept in game theory, is the best strategy for any player, no matter what strategies the other players choose. Due to the above characteristic, researchers have paid great attention to approaching the Nash equilibrium^[46, 47].

Tree search methods have long been the mainstream for turn-based games. Typical methods such as min-max search, MCTS, and CFR are classical algorithms that can approach Nash equilibrium, so those techniques are widely utilized in games such as chess and limit poker. However, when facing complex environments such as Go and HUNL, the calculation of Nash equilibrium is untraceable because of the huge game tree complexity. To handle this problem, properly restricting the depth and width of the game tree becomes a very important strategy where deep learning can be used. For example, the AlphaGo series train policy and value networks to pay more attention to valuable nodes to be expanded and evaluate nodes expanded, respectively.

In complex real-time video games, we cannot easily draw lessons from tree search methods because of challenges such as long time horizons and complex action

space. Fictitious self-play^[48] provides an evolutionary strategy for agent learning, which can approach the Nash equilibrium in certain types of games, such as two players zero-sum games and potential games, and exceptions, such as multiple players zero-sum games can not guarantee convergence. However, the computation of fictitious self-play for a complex game is high due to the best response calculation and average strategies updating, so researchers develop various self-play or population-play strategies and use distributed reinforcement learning to learn each generation of agents. Even though a theoretical guarantee of Nash equilibrium is absent, professional-level AIs can be trained by properly overcoming game-theoretic challenges. For example, OpenAI Five plays against itself for 80% of the games and against past generations for 20% of the games by their winning rate against the current version. AlphaStar designs three types of agents to enhance self-play, where each type of agent performs confrontation with certain opponents to gradually improve the performance of the main agents without desperation or just learning a narrow of policies.

7.2 How to become general technology?

Considering real-world games are mostly real-time with many decisions to be made, and players usually form their decisions not in an iterative manner, tree search-based methods are more challenging to implement in very complex games. However, self-play or population-play with distributed learning can be a general solution due to its simple implementation and performance guarantee, such as the success of AlphaStar and OpenAI Five. Generally, there are three steps to train an AI based on this technique, as shown in Fig. 6.

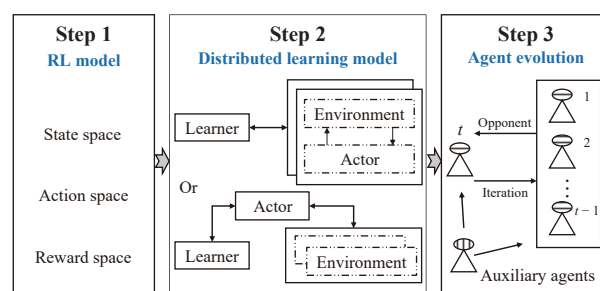


Fig. 6 Steps for a general technology to train AIs

Firstly, the task can be properly modeled as a reinforcement learning framework, which consists of several key factors. Usually, determining the state space and action space is the most important part. The former provides information for neural network input, which should be rich enough for a suitable decision and lean enough to reduce the computation burden. The latter reflects how to drive environment transfer. Too complex of action representation will increase learning difficulty, but too simple design will make the agent unable to reach the

professional level due to action limitations. What's more, when performing reinforcement learning, how to design reward space is another key factor because it is the task signal to learn each generation of agents. The too-sparse reward on a long time horizon game will greatly increase learning difficulty, but designing immediate rewards to guide agents pursuing task rewards needs a lot of human experience.

With the above factors, one can design or adopt reinforcement learning algorithms such as Q learning, advantage actor critic^[49], and proximal policy optimization^[50] for agent learning. Usually, deep neural networks are specially designed to transform input state information to output action, e.g., auto-regressive policy to deal with structured and combinatorial action space in AlphaStar. To accelerate reinforcement learning, distributed learning should be carefully designed based on the model inference cost to drive rollout, the communication cost among machines to transfer data, and most importantly, the machine configuration, such as the GPU and CPU ability. For example, when the model is relatively small and the inference cost is low, one can choose a distributed framework like in FTW. Nowadays, Tensorflow⁴, Pytorch⁵, and several tools such as Ray^[51] and Horovod^[52] can easily achieve multiple machines distributed learning with minimal code changes compared to that in a single machine^[53].

Finally, since each generation of agents can be trained based on distributed reinforcement learning, the last step is to design a self-play or population-play based mechanism for agent evolution. Currently, many heuristic approaches have been developed. For example, AlphaStar uses three types of agents, with each type selecting different opponents, based on which all types of agents evolve to make the main agent stronger. Overall, previous evolution strategies for self-play or population-play are mostly heuristic, and one can design strategies based on the game faced to improve the agent's ability.

8 Challenges and future trends

Even though big progress has been made in human-computer gaming, current techniques have at least one of three limitations. Firstly, most AIs are designed for a specific human-computer game or a map of a specific game, and the AIs learned are not able to be used even for different maps of a game. Moreover, not enough experiments are performed to validate the AI's ability when a disturbance is brought into the game. Secondly, training the above AIs requires a large number of computation resources, which will be elaborated on in the following subsection. Due to the huge hardware resource threshold, only a limited number of organizations are capable of training high-level AIs, which will obstruct the majority

of scientific research from an in-depth study of the problem. Thirdly, most AIs are evaluated based on their winning ability against limited professional human players, and a claim of reaching the expert level may be a little exaggerated. In Sections 8.1–8.4, we will show the potential directions and challenges faced by the above limitations.

8.1 Big model

Nowadays, big models, especially pretrained big models, are emerging from natural language processing to the computer version, from the single modality to multiple modalities^[54, 55]. These models have proven great potential for downstream tasks even in zero-shot settings, which is a big step towards exploring artificial general intelligence. For example, OpenAI developed generative pre-trained transformer 3 (GPT-3)^[56], which has more than 175 billion parameters and displays promising performance in various language-related tasks. However, big models in games are largely absent, and current models for complex games are much smaller than those big ones. As shown in Table 3, AlphaStar and OpenAI Five only have 139 million and 159 million parameters, respectively.

Table 3 Parameter sizes of current AIs and pretrained models

Models	Parameter size
JueWu ^[15]	17 million
Commander ^[21]	49 million
AlphaStar ^[10]	139 million
OpenAI Five ^[9]	159 million
GPT-3 ^[56]	175 billion
Megatron-Turing NLG ⁶	530 billion
M6-10T ^[57]	10 trillion

Considering that the big model is a relatively good exploration for artificial general intelligence, how to design and train a big model for AI in human-computer gaming may provide a solution for those sequential decision-making problems. To make such an attempt, we think several problems should be carefully considered.

Firstly, unlike natural language processing problems, tasks for games are very different, so how to make clear training goals is the key step for a big model. For example, in StarCraft, players need to build a force with at most 200 units to fight against enemies, but in Dota2, five heroes work together to defeat another five heroes. Even though distinct actions or skills are required for different games, the mechanism of playing a game is similar, i.e., extract useful information from image streams and

⁴ <https://www.tensorflow.org/>

⁵ <https://pytorch.org/>

⁶ <https://www.microsoft.com/en-us/research/blog/using-deeppsp-ee-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/>

make a decision based on the current situation. Therefore, a possible breakthrough point is to learn the high-level strategic situation to provide information for decisions. Note that other goals for training big models are welcome as long as they can provide general and useful information for making decisions.

Secondly, since some games are hard and some games are easy, how to design a suitable training mechanism is difficult. It should handle various kinds of games and make sure the learning does not degenerate, e.g., not forgetting the representation ability^[22]. Continual learning provides a tool for such problems^[58, 59], but there are still several issues that need to be carefully handled. Since training a high-level game AI is an evolutionary process that needs self-play or other iterative learning, how to properly embed evolution into the above learning mechanism is a problem that has never been faced. On the other hand, different games share similar characteristics to some extent, establishing the connection between them when performing training is a key factor to reduce complexity and promote performance.

8.2 Low resources AI

To train professional-level AIs for complex environments, large computational resources are usually required. As shown in Table 4, we can find a huge resource devoted to training an AI.

Table 4 Computational resources for professional AIs

AIs	Resources
AlphaZero	5 000 v1 TPUs and 16 v2 TPUs for 13 days
Libratus	25 million core hours
OpenAI Five	770 ± 50 PFlops/s-day for 10 months
AlphaStar	192 v3 + 12 128 core TPUs, 1 800 CPUs for 44 days

One question naturally arises whether it is possible to train a professional-level AI with limited resources. The intuitive idea is to bring in more human knowledge to assist learning^[60, 61]. For example, incorporating prior knowledge as constraints or loss functions can improve conventional machine learning algorithms. Since current breakthroughs in games are mostly relying on reinforcement learning which is sample inefficient, how to achieve sample efficient reinforcement learning based on human knowledge is a future direction^[62, 63].

On the other hand, training a professional agent is usually an evolution process, which iteratively learns hundreds of models. For example, in AlphaStar, almost 900 different players are created, with each one maintaining a specific kind of task. Therefore, reducing this iteration seems to be an effective medium for reducing computational resources. Current approaches, mainly based on self-play, are mostly heuristic by selecting suitable oppon-

ents for the current generation of agents. If theoretical and easy-to-calculate evolution strategies are developed, it will be a key step for low resources AIs.

8.3 AI evaluation

Most games in the real world are in-transitive, i.e., transitive and in-transitive parts co-exist^[25]. The in-transitive characteristic makes the precise evaluation of an agent a difficult problem. Current human-computer gaming usually utilizes a winning rate (against professional human players) based evaluation criteria, as shown in Table 5. However, such evaluation is relatively rough, especially under limited tests for in-transitivity games.

Table 5 Evaluation of typical AIs

AIs	Resources
AlphaGo Zero	Previous AlphaGo series ⁷
Suphx	99.99% of all the officially ranked human players
Libratus	Four top human specialist professionals
OpenAI Five	Professional teams with world champions OG
AlphaStar	99.8% of ranked human players
JueWu	95.2% win rate against professional players

Theoretically, Nash equilibrium is a relatively conservative solution due to not considering the weakness of opponents^[64–66]. Still, it is the best solution for any kind of opponent in non-cooperative games. Accordingly, how to evaluate the distance between the learned solution and the Nash equilibrium solution is an important problem. Moreover, it may help us figure out if AlphaZero reaches the Nash equilibrium and can not be beaten by any humans.

On the other hand, current ranking methods for humans and AIs are based on their battle records, such as Elo^[29, 67, 68]. However, under in-transitive games, such calculation is inexact. Moreover, the win rate is just one of the evaluation metrics, and it may not be enough to reflect all the aspects of an agent. Accordingly, how to develop systematic evaluation criteria for most games can be an important and open problem.

8.4 New challenging games

After the breakthrough of AlphaStar, researchers are looking for new games for advancing decision-making intelligence, e.g., football. In our opinion, current games with great progress are mostly symmetrical in ability. Even though games like StarCraft and Dota2 look asymmetric because there are three distinct races with differ-

⁷ Including AlphaGo Master, a previous version of AlphaGo Zero that defeated strongest human professional players by 60 – 0 in online games.

ent forces in StarCraft and plenty of heroes with diverse skills in Dota2, those games share a common characteristic of balance for different choices. This is important for games that are popular for humans, e.g., an eSports game.

On the contrary, the real world is full of asymmetric games, and it is almost unable to find a strictly symmetrical game in our surroundings^[69]. Therefore, it may be a good direction to design asymmetrical games (mostly asymmetrical in ability) to develop decision-making intelligence for real world problems. Currently, there are few environments for asymmetrical games, and researchers are less focused on developing techniques for those kinds of testbeds^[70]. We argue that previous training frameworks, especially self-play with distributed learning, can not deal with such scenarios because a two-player asymmetrical game has very different strategies for different sides, and self-play based mechanisms may not work well.

Wargame⁸ is a popular confrontation game, as shown in Fig. 7, where two players (red and blue) each control a collection of combat units fighting against each other^[22]. Based on several settings of Wargame, two sides are asymmetrical in ability, and usually the power of the red one is weaker than that of the blue one. Considering Wargame is a complex game facing imperfect information, long time horizon, in-transitive strategy, multi-agent cooperation, and asymmetrical in ability, it may be a new testbed for AI in human-computer gaming.

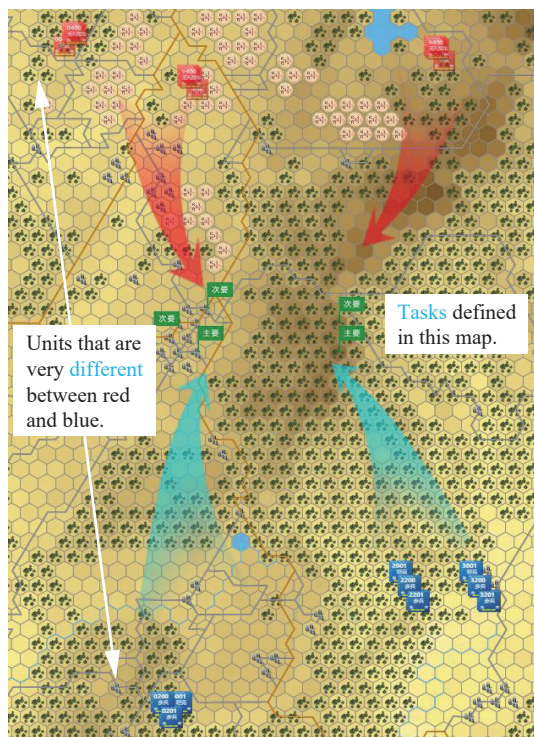


Fig. 7 A screenshot of wargame

⁸ Come from <http://wargame.ia.ac.cn/main>

9 Conclusions

In this paper, we have summarized and compared techniques of current breakthroughs of AIs in human-computer gaming, covering board games, card games, FPS games, and RTS games. The main difficulties among different kinds of games are illustrated, and learning frameworks of representative human-computer gaming AIs are elaborated with detailed comparisons. Based on the comparison, we illustrate two mainstream frameworks used for developing professional-level AIs and how to use one of them to be a general technology for developing AIs. More importantly, we summarize the main limitations of current AIs, trying to propose future directions along with the challenges faced in the field. Through this survey, we hope that beginners can quickly become familiar with the techniques, challenges, and opportunities in this exciting field, and researchers on the way can be inspired for deeper study.

Acknowledgements

This work was supported by National Natural Science Foundation of China (No.61906197).

Open Access

This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made.

The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- [1] N. E. Cagiltay. Teaching software engineering by means of computer-game development: Challenges and opportunities. *British Journal of Educational Technology*, vol.38, no.3, pp.405–415, 2007. DOI: [10.1111/j.1467-8535.2007.00705.x](https://doi.org/10.1111/j.1467-8535.2007.00705.x).
- [2] R. J. Crouser, R. Chang. An affordance-based framework for human computation and human-computer collaboration. *IEEE Transactions on Visualization and Computer Graphics*, vol.18, no.12, pp.2859–2868, 2012. DOI: [10.1109/TVCG.2012.195](https://doi.org/10.1109/TVCG.2012.195).
- [3] A. P. Saygin, I. Cicekli, V. Akman. Turing test: 50 years

- later. *Minds and Machines*, vol.10, no.4, pp.463–518, 2000. DOI: [10.1023/A:1011288000451](https://doi.org/10.1023/A:1011288000451).
- [4] J. Schaeffer, R. Lake, P. Lu, M. Bryant. Chinook the world man-machine checkers champion. *AI Magazine*, vol.17, no.1, Article number 21, 1996. DOI: [10.1609/aimag.v17i1.1208](https://doi.org/10.1609/aimag.v17i1.1208).
 - [5] M. Campbell, A. J. Jr. Hoane, F. H. Hsu. Deep Blue. *Artificial Intelligence*, vol.134, no.1–2, pp.57–83, 2002. DOI: [10.1016/S0004-3702\(01\)00129-1](https://doi.org/10.1016/S0004-3702(01)00129-1).
 - [6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, vol.518, no.7540, pp.529–533, 2015. DOI: [10.1038/nature14236](https://doi.org/10.1038/nature14236).
 - [7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, vol.529, no.7587, pp.484–489, 2016. DOI: [10.1038/nature16961](https://doi.org/10.1038/nature16961).
 - [8] N. Brown, T. Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, vol.359, no.6374, pp.418–424, 2018. DOI: [10.1126/science.aao1733](https://doi.org/10.1126/science.aao1733).
 - [9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. D biak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. J zefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, S. S. Zhang. Dota 2 with large scale deep reinforcement learning. [Online], Available: <https://arxiv.org/abs/1912.06680v1>, 2019.
 - [10] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Y. Wang, T. Pfaff, Y. H. Wu, R. Ring, D. Yogatama, D. W nsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, vol.575, no.7782, pp.350–354, 2019. DOI: [10.1038/s41586-019-1724-z](https://doi.org/10.1038/s41586-019-1724-z).
 - [11] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, C. Blundell. Agent57: Outperforming the Atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning*, pp.507–517, 2020.
 - [12] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, I. Mordatch. Emergent tool use from multi-agent autocurricula. [Online], Available: <https://arxiv.org/abs/1909.07528>, 2019.
 - [13] N. Brown, T. Sandholm. Superhuman AI for multiplayer poker. *Science*, vol.365, no.6456, pp.885–890, 2019. DOI: [10.1126/science.aay2400](https://doi.org/10.1126/science.aay2400).
 - [14] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. T. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, D. Hassabis. Mastering the game of go without human knowledge. *Nature*, vol.550, no.7676, pp.354–359, 2017. DOI: [10.1038/nature24270](https://doi.org/10.1038/nature24270).
 - [15] D. H. Ye, G. B. Chen, W. Zhang, S. Chen, B. Yuan, B. Liu, J. Chen, Z. Liu, F. H. Qiu, H. S. Yu, Y. Y. T. Yin, B. Shi, L. Wang, T. F. Shi, Q. Fu, W. Yang, L. X. Huang, W. Liu. Towards playing full MOBA games with deep reinforcement learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Vancouver, Canada, Article number 53, 2020.
 - [16] J. J. Li, S. Koyamada, Q. W. Ye, G. Q. Liu, C. Wang, R. H. Yang, L. Zhao, T. Qin, T. Y. Liu, H. W. Hon. Suphx: Mastering mahjong with deep reinforcement learning. [Online], Available: <https://arxiv.org/abs/2003.13590v2>, 2020.
 - [17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, vol.362, no.6419, pp.1140–1144, 2018. DOI: [10.1126/science.aar6404](https://doi.org/10.1126/science.aar6404).
 - [18] M. Morav ik, M. Schmid, N. Burch, V. Lis y, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, vol.356, no.6337, pp.508–513, 2017. DOI: [10.1126/science.aam6960](https://doi.org/10.1126/science.aam6960).
 - [19] D. C. Zha, J. R. Xie, W. Y. Ma, S. Zhang, X. R. Lian, X. Hu, J. Liu. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, pp.12333–12344, 2021.
 - [20] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Casta eda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, T. Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, vol.364, no.6443, pp.859–865, 2019. DOI: [10.1126/science.aau6249](https://doi.org/10.1126/science.aau6249).
 - [21] X. J. Wang, J. X. Song, P. H. Qi, P. Peng, Z. K. Tang, W. Zhang, W. M. Li, X. J. Pi, J. J. He, C. Gao, H. T. Long, Q. Yuan. SCC: An efficient deep reinforcement learning agent mastering the game of StarCraft II. In *Proceedings of the 38th International Conference on Machine Learning*, pp.10905–10915, 2021.
 - [22] Q. Y. Yin, M. J. Zhao, W. C. Ni, J. G. Zhang, K. Q. Huang. Intelligent decision making technology and challenge of wargame. *Acta Automatica Sinica*, vol.48, no.9, pp.1–15, 2022. DOI: [10.16383/j.aas.c210547](https://doi.org/10.16383/j.aas.c210547). (in Chinese)
 - [23] N. Brown, T. Sandholm. Safe and nested subgame solving for imperfect-information games. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, Long Beach, USA, pp.689–699, 2017.
 - [24] N. Burch. Time and Space: Why Imperfect Information Games are Hard, Ph.D. dissertation, Department of Computing Science, University of Alberta, Edmonton, Canada, 2017.
 - [25] R. Sanjaya, J. Wang, Y. D. Yang. Measuring the non-transitivity in chess. *Algorithms*, vol.15, no.5, Article number 152, 2022. DOI: [10.3390/a15050152](https://doi.org/10.3390/a15050152).

- [26] W. M. Czarnecki, G. Gidel, B. Tracey, K. Tuyls, S. Omidshafiei, D. Balduzzi, M. Jaderberg. Real world games look like spinning tops. In *Proceedings of the 34th Conference on Neural Information Processing Systems*, Vancouver, Canada, pp. 17443–17454, 2020.
- [27] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012. DOI: [10.1109/TCIAIG.2012.2186810](https://doi.org/10.1109/TCIAIG.2012.2186810).
- [28] L. Schaefers, M. Platzner. Distributed Monte Carlo tree search: A novel technique and its application to computer go. *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 361–374, 2015. DOI: [10.1109/TCIAIG.2014.2346997](https://doi.org/10.1109/TCIAIG.2014.2346997).
- [29] A. E. Elo. *The Rating of Chess Players, Past & Present*, New York, USA: Arco Pub, 1978.
- [30] M. Zinkevich, M. Johanson, M. Bowling, C. Piccione. Regret minimization in games with incomplete information. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, Vancouver, Canada, pp. 1729–1736, 2007.
- [31] M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling. Monte Carlo sampling for regret minimization in extensive games. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, Vancouver, Canada, pp. 1078–1086, 2009.
- [32] N. Burch, M. Johanson, M. Bowling. Solving imperfect information games using decomposition. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, Québec City, Canada, pp. 602–608, 2013.
- [33] Y. Li, D. Xu. Skill learning for robotic insertion based on one-shot demonstration and reinforcement learning. *International Journal of Automation and Computing*, vol. 18, no. 3, pp. 457–467, 2021. DOI: [10.1007/s11633-021-1290-3](https://doi.org/10.1007/s11633-021-1290-3).
- [34] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. DOI: [10.1109/MSP.2017.2743240](https://doi.org/10.1109/MSP.2017.2743240).
- [35] Y. Yu. Towards sample efficient reinforcement learning. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, pp. 5739–5743, 2018.
- [36] T. M. Moerland, J. Broekens, C. M. Jonker. Model-based reinforcement learning: A survey. [Online], Available: <https://arxiv.org/abs/2006.16712>, 2020.
- [37] A. J. Baruah, S. Baruah. Data augmentation and deep neuro-fuzzy network for student performance prediction with MapReduce framework. *International Journal of Automation and Computing*, vol. 18, no. 6, pp. 981–992, 2021. DOI: [10.1007/s11633-021-1312-1](https://doi.org/10.1007/s11633-021-1312-1).
- [38] M. R. Samsami, H. Alimadad. Distributed deep reinforcement learning: An overview. [Online], Available: <https://arxiv.org/abs/2011.11012>, 2020. DOI: [10.48550/arxiv.2011.11012](https://doi.org/10.48550/arxiv.2011.11012).
- [39] T. Y. Chen, K. Q. Zhang, G. B. Giannakis, T. Başar. Communication-efficient distributed reinforcement learning. [Online], Available: <https://arxiv.org/abs/1812.03239>, 2018.
- [40] A. Nair, P. Srinivasan, S. Blackwell, C. Alciçek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, D. Silver. Massively parallel methods for deep reinforcement learning. [Online], Available: <https://arxiv.org/abs/1507.04296v2>, 2015.
- [41] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, K. Kavukcuoglu. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, pp. 1406–1415, 2018.
- [42] L. Espeholt, R. Marinier, P. Stanczyk, K. Wang, M. Michalski. SEED RL: Scalable and efficient Deep-RL with accelerated central inference. In *Proceedings of the 8th International Conference on Learning Representations*, Addis Ababa, Ethiopia, pp. 1–20, 2020.
- [43] M. W. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, N. Momchev, D. Sinopalnikov, P. Stańczyk, S. Ramos, A. Raichuk, D. Vincent, L. Hussenot, R. Dadashi, G. Dulac-Arnold, M. Orsini, A. Jacq, J. Ferret, N. Vieillard, S. K. S. Ghasemipour, S. Girgin, O. Pietquin, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Friesen, R. Haroun, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. Le Paine, S. Srinivasan, A. Cowie, Z. Y. Wang, B. Piot, N. de Freitas. Acme: A research framework for distributed reinforcement learning. [Online], Available: <https://arxiv.org/abs/2006.00979>, 2020.
- [44] W. J. Hong, P. Yang, K. Tang. Evolutionary computation for large-scale multi-objective optimization: A decade of progresses. *International Journal of Automation and Computing*, vol. 18, no. 2, pp. 155–169, 2021. DOI: [10.1007/s11633-020-1253-0](https://doi.org/10.1007/s11633-020-1253-0).
- [45] C. Daskalakis, P. W. Goldberg, C. H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing*, vol. 39, no. 1, pp. 195–259, 2009. DOI: [10.1137/070699652](https://doi.org/10.1137/070699652).
- [46] N. Brown, A. Bakhtin, A. Lerer, Q. C. Gong. Combining deep reinforcement learning and search for imperfect-information games. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, Vancouver, Canada, Article number 1431, 2020.
- [47] Y. F. Lv, X. M. Ren. Approximate Nash solutions for multiplayer mixed-zero-sum game with reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 12, pp. 2739–2750, 2019. DOI: [10.1109/TSMC.2018.2861826](https://doi.org/10.1109/TSMC.2018.2861826).
- [48] J. Heinrich, M. Lanctot, D. Silver. Fictitious self-play in extensive-form games. In *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, pp. 805–813, 2015.
- [49] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, New York, USA, pp. 1928–1937, 2016.
- [50] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. Proximal policy optimization algorithms. [Online], Available: <https://arxiv.org/abs/1707.06347>, 2017.
- [51] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw,

- E. Liang, M. Elibol, Z. H. Yang, W. Paul, M. I. Jordan, I. Stoica. Ray: A distributed framework for emerging AI applications. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*, Carlsbad, USA, pp. 561–577, 2018.
- [52] A. Sergeev, M. Del Balso. Horovod: Fast and easy distributed deep learning in TensorFlow. [Online], Available: <https://arxiv.org/abs/1802.05799>, 2018.
- [53] T. Kurth, M. Smorkalov, P. Mendygral, S. Sridharan, A. Mathuriya. TensorFlow at scale: Performance and productivity analysis of distributed training with Horovod, MLML, and Cray PE ML. *Concurrency and Computation: Practice and Experience*, vol. 31, no. 16, Article number e4989, 2019. DOI: [10.1002/cpe.4989](https://doi.org/10.1002/cpe.4989).
- [54] X. P. Qiu, T. X. Sun, Y. G. Xu, Y. F. Shao, N. Dai, X. J. Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, vol. 63, no. 10, pp. 1872–1897, 2020. DOI: [10.1007/s11431-020-1647-3](https://doi.org/10.1007/s11431-020-1647-3).
- [55] J. Y. Lin, R. Men, A. Yang, C. Zhou, M. Ding, Y. C. Zhang, P. Wang, A. Wang, L. Jiang, X. Y. Jia, J. Zhang, J. W. Zhang, X. Zou, Z. K. Li, X. D. Deng, J. Liu, J. B. Xue, H. L. Zhou, J. X. Ma, J. Yu, Y. Li, W. Lin, J. R. Zhou, J. Tang, H. X. Yang. M6: A Chinese multimodal pretrainer. [Online], Available: <https://arxiv.org/abs/2103.00823>, 2021.
- [56] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei. Language models are few-shot learners. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems*, Vancouver, Canada, pp. 1877–1901, 2020.
- [57] J. Y. Lin, A. Yang, J. Z. Bai, C. Zhou, L. Jiang, X. Y. Jia, A. Wang, J. Zhang, Y. Li, W. Lin, J. R. Zhou, H. X. Yang. M6-10T: A sharing-delinking paradigm for efficient multi-trillion parameter pretraining. [Online], Available: <https://arxiv.org/abs/2110.03888>, 2021.
- [58] M. Biesialska, K. Biesialska, M. R. Costa-Jussà. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain, pp. 6523–6541, 2020. DOI: [10.18653/v1/2020.coling-main.574](https://doi.org/10.18653/v1/2020.coling-main.574).
- [59] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3366–3385, 2022. DOI: [10.1109/TPAMI.2021.3057446](https://doi.org/10.1109/TPAMI.2021.3057446).
- [60] L. V. Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, M. Walczak, J. Pfrommer, A. Pick, R. Ramamurthy, J. Garcke, C. Bauckhage, J. Schuecker. Informed machine learning – A taxonomy and survey of integrating prior knowledge into learning systems. *IEEE Transactions on Knowledge and Data Engineering*, to be published. DOI: [10.1109/TKDE.2021.3079836](https://doi.org/10.1109/TKDE.2021.3079836).
- [61] Y. Wu, J. H. Li, Y. Z. Yuan, A. K. Qin, Q. G. Miao, M. G. Gong. Commonality autoencoder: Learning common features for change detection from heterogeneous images. *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4257–4270, 2022. DOI: [10.1109/TNNLS.2021.3056238](https://doi.org/10.1109/TNNLS.2021.3056238).
- [62] M. K. Titsias, S. Nikoloutsopoulos. Bayesian transfer reinforcement learning with prior knowledge rules. [Online], Available: <https://arxiv.org/abs/1810.00468>, 2018.
- [63] N. Bougie, L. K. Cheng, R. Ichise. Combining deep reinforcement learning with prior knowledge and reasoning. *ACM SIGAPP Applied Computing Review*, vol. 18, no. 2, pp. 33–45, 2018. DOI: [10.1145/3243064.3243067](https://doi.org/10.1145/3243064.3243067).
- [64] M. C. Shen, J. P. How. Robust opponent modeling via adversarial ensemble reinforcement learning. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling*, Guangzhou, China, pp. 578–587, 2021.
- [65] M. J. Kim, K. J. Kim. Opponent modeling based on action table for MCTS-based fighting game AI. In *Proceedings of IEEE Conference on Computational Intelligence and Games*, New York, USA, pp. 178–180, 2017. DOI: [10.1109/CIG.2017.8080432](https://doi.org/10.1109/CIG.2017.8080432).
- [66] F. B. Von Der Osten, M. Kirley, T. Miller. The minds of many: Opponent modelling in a stochastic game. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, Melbourne, Australia, pp. 3845–3851, 2017.
- [67] D. Balduzzi, K. Tuyls, J. Perolat, T. Graepel. Re-evaluating evaluation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, Montréal, Canada, pp. 3272–3283, 2018.
- [68] S. Omidshafiei, C. Papadimitriou, G. Piliouras, K. Tuyls, M. Rowland, J. B. Lespiau, W. M. Czarnecki, M. Lanctot, J. Perolat, R. Munos. α -rank: Multi-agent evaluation by evolution. *Scientific Reports*, vol. 9, no. 1, Article number 9937, 2019. DOI: [10.1038/s41598-019-45619-9](https://doi.org/10.1038/s41598-019-45619-9).
- [69] K. Tuyls, J. Pérolat, M. Lanctot, G. Ostrovski, R. Savani, J. Z. Leibo, T. Ord, T. Graepel, S. Legg. Symmetric decomposition of asymmetric games. *Scientific Reports*, vol. 8, no. 1, Article number 1015, 2018. DOI: [10.1038/s41598-018-19194-4](https://doi.org/10.1038/s41598-018-19194-4).
- [70] H. T. Jia, Y. J. Hu, Y. F. Chen, C. X. Ren, T. J. Lv, C. J. Fan, C. J. Zhang. Fever basketball: A complex, flexible, and asynchronous sports game environment for multi-agent reinforcement learning. [Online], Available: <https://arxiv.org/abs/2012.03204>, 2020.



Qi-Yue Yin received the Ph.D. degree in pattern recognition and intelligent system from National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences (CASIA), China in 2017. He is currently an associate professor at Center for Research on Intelligent System and Engineering, CASIA, China.

His research interests include machine learning, pattern recognition and artificial intelligence on games.

E-mail: qyyin@nlpr.ia.ac.cn

ORCID iD: 0000-0002-3442-6275



Jun Yang received the Ph.D. degree in control science and engineering from Tsinghua University, China in 2011. He is currently a lecturer with Department of Automation, Tsinghua University, China.

His research interests include multi-agent reinforcement learning and game theory.

E-mail: yangjun603@tsinghua.edu.cn

(Corresponding author)

ORCID iD: 0000-0002-9386-5825



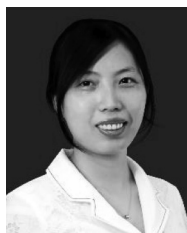
Kai-Qi Huang received the Ph.D. degree in communication and information processing from Southeast University, China in 2004. He is currently a professor at Center for Research on Intelligent System and Engineering, CASIA, China.

His research interests include visual surveillance, image understanding, pattern recognition, human-computer gaming and

biological based vision.

E-mail: kqhuang@nlpr.ia.ac.cn (Corresponding author)

ORCID iD: 000-0002-2677-9273



Mei-Jing Zhao received the Ph.D. degree in pattern recognition and intelligent system from Integrated Information System Research Center, CASIA, China in 2016. She is currently an associate professor at Center for Research on Intelligent System and Engineering, CASIA, China.

Her research interests include semantic information processing, knowledge representation and reasoning.

E-mail: meijing.zhao@ia.ac.cn



Wan-Cheng Ni received the Ph.D. degree in contemporary integrated manufacturing systems from Department of Automation, Tsinghua University, China in 2007. She is currently a professor at Center for Research on Intelligent System and Engineering, CASIA, China.

Her research interests include information processing and knowledge discovery,

group intelligent decision-making platform and evaluation.

E-mail: wancheng.ni@ia.ac.cn



Bin Liang received the Ph.D. degree in precision instruments and mechatronics from Tsinghua University, China in 1994. He is currently a professor with Department of Automation, Tsinghua University, China.

His research interests include artificial intelligence, anomaly detection, space robotics, and fault-tolerant control.

E-mail: bliang@tsinghua.edu.cn



Yan Huang received the Ph.D. degree in pattern recognition and intelligent system from National Laboratory of Pattern Recognition (NLPR), CASIA, China in 2017. He is currently an associate professor at NLPR, CASIA, China.

His research interests include visual language understanding and video analysis.

E-mail: yhuang@nlpr.ia.ac.cn



Shu Wu received the Ph.D. degree in computer science from University of Sherbrooke, Canada in 2012. He is currently an associate professor at National Laboratory of Pattern Recognition, CASIA, China.

His research interests include data mining, network content analysis and security.

E-mail: shu.wu@nlpr.ia.ac.cn



Liang Wang received the Ph.D. degree in pattern recognition and intelligent system from National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences, China in 2004. He is currently a professor at NLPR, CASIA, China.

His research interests include computer vision, pattern recognition, machine learning and data mining.

E-mail: wangliang@nlpr.ia.ac.cn