TiKick: Towards Playing Multi-agent Football Full Games from Single-agent Demonstrations

Shiyu Huang[*]
Tsinghua University
Beijing, China
hsy17@mails.tsinghua.edu.cn

Wenze Chen[*]
Tsinghua University
Beijing, China
cwz19@mails.tsinghua.edu.cn

Longfei Zhang
National University of Defense Technology
Changsha, China
zhanglongfei@nudt.edu.cn

Ziyang Li
Tencent AI Lab
Shenzhen, China
tzeyangli@tencent.com

Fengming Zhu
Tencent AI Lab
Shenzhen, China
fridazhu@tencent.com

Deheng Ye
Tencent AI Lab
Shenzhen, China
dericye@tencent.com

Ting Chen
Tsinghua University
Beijing, China
tingchen@tsinghua.edu.cn

Jun Zhu
Tsinghua University
Beijing, China
dcszj@tsinghua.edu.cn

Abstract

Deep reinforcement learning (DRL) has achieved super-human performance on complex video games (e.g., StarCraft II and Dota II). However, current DRL systems still suffer from challenges of multi-agent coordination, sparse rewards, stochastic environments, etc. In seeking to address these challenges, we employ a football video game, e.g., Google Research Football (GRF), as our testbed and develop an end-to-end learning-based AI system (denoted as TiKick[2][3]) to complete this challenging task. In this work, we first generated a large replay dataset from the self-playing of single-agent experts, which are obtained from league training. We then developed a distributed learning system and new offline algorithms to learn a powerful multi-agent AI from the fixed single-agent dataset. To the best of our knowledge, Tikick is the first learning-based AI system that can take over the multi-agent Google Research Football full game, while previous work could either control a single agent or experiment on toy academic scenarios. Extensive experiments further show that our pre-trained model can accelerate the training process of the modern multi-agent algorithm and our method achieves state-of-the-art performances on various academic scenarios.

[*]Equal contribution
[2]Codes can be found at https://github.com/TARTRL/TiKick.
[3]Videos available at https://sites.google.com/view/tikick.

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2021.

1 Introduction

Deep reinforcement learning (DRL) has shown great success in many video games, including the Atari games (Mnih et al., 2013), StarCraft II (Vinyals et al., 2019), Dota II (Berner et al., 2019), etc. However, current DRL systems still suffer from challenges of multi-agent coordination (Rashid et al., 2018; Mahajan et al., 2019; Yu et al., 2021), sparse rewards (Taiga et al., 2019; Zhang et al., 2020), stochastic environments (Kurach et al., 2019; Team et al., 2021), etc. In seeking to address these challenges, we employ a football video game, e.g., Google Research Football (GRF) (Kurach et al., 2019), as our testbed. Even much work has been done recently (Kurach et al., 2019; Li et al., 2021; Liu et al., 2021b), there remain many problems in building agents for the GRF: (1) **Multiple Players**: In the GRF, there are both cooperative and competitive players. For cooperative players, the joint action space is very huge, thus it is hard to build a single agent to control all the players. Moreover, competitive players mean that the opponents are not fixed, thus the agents should be adapted to various opponents. (2) **Sparse Rewards**: The goal of the football game is to maximize the goal score, which can only be obtained after a long time of the perfect decision process. And it is almost impossible to receive a positive reward when starting from random agents. (3) **Stochastic Environments**: The GRF introduces stochasticity into the environment, which means the outcome after taking certain actions is not deterministic. This can improve the robustness of the trained agents but also increase the training difficulties. To address the aforementioned issues, we develop an end-to-end learning-based AI system (denoted as TiKick) to complete this challenging task. In this work, we first generated a large replay dataset from the self-playing of single-agent experts, which are obtained from league training. We then developed a distributed learning system and new offline algorithms to learn a powerful multi-agent AI from the fixed single-agent dataset. To the best of our knowledge, Tikick is the first learning-based AI system that can take over the multi-agent Google Research Football full game, while previous work could either control a single agent or experiment on toy academic scenarios. Extensive experiments further show that our pre-trained model can accelerate the training process of the modern multi-agent algorithm and our method can achieve state-of-the-art performances on various academic scenarios.

2 Background

Many researchers have developed agents for other football games (e.g., *RoboCup Soccer Simulator* (Kitano et al., 1997) and the *DeepMind MuJoCo Multi-Agent Soccer Environment* (Liu et al., 2019, 2021a)). Different from Google Research Football, these environments focus more on low-level control of a physics simulation of robots, while GRF focuses on high-level actions. In Google Research Football Competition 2020 (Google, 2020), an RL-based agent, named WeKick (Ziyang Li, 2020), took first place. WeKick utilized imitation learning, the multi-head value trick, and distributed league training to achieve the top performance on the GRF. However, it is a single-agent AI that can not be extended to multi-agent control. And Liu et al. (2021b) also proposed a new algorithm for the GRF with single-agent control. Instead, our work tries to build a powerful game AI for multi-agent control on the GRF. More recently, Li et al. (2021) proposed a value-based multi-agent algorithm (i.e., CDS) for GRF with mutual information maximization between agents. However, they only conducted experiments on toy academic scenarios. Compared with CDS, our AI system can achieve much better performances and lower sample complexity on these academic scenarios without the mutual information maximization process, and our method can even play the GRF full game.

3 Methodology

In this section, we will introduce the TiKick algorithm in detail, including our observation, model design, and our proposed offline reinforcement learning method.

3.1 Observation and Model Design

Design the observation is the first step for building a DRL model. The Google Research Football simulator originally provide three types of observations. The first type is the pixel-level representation, which consists of a 1280×720 RGB image corresponding to the rendered screen. The first type is the *Super Mini Map* (SMM), which consists of four 72×96 matrices encoding information about the current situation. The third type is the vector representation, which provides a compact

encoding and consists of a 115-dimensional vector. The pixel-level and SMM representations need more burdensome deep models to extract hidden features, which results in lower training speed and higher memory consumption. Thus, we use the vector representation as the input of our deep model. Furthermore, we extend the standard 115-dimensional vector with auxiliary features, such as relative poses among teammates and opponents, offside flags to mark potential offside teammates, etc. Finally, we obtain a 268-dimensional vector as the input. More details about the observation can be found in the Appendix.

To extract hidden features from the raw input, we build a deep model with four fully connected layers and GRU cells (Cho et al., 2014). All the hidden layers are followed by a ReLU, except for the last output layer and all the hidden sizes are set to 256. The learning rate is set to $1e-4$ and is fixed during the training. Network parameters are initialized with the orthogonal matrix (Saxe, McClelland, and Ganguli, 2013) and Adam optimizer (Kingma and Ba, 2014) is used for the parameter update. More details about the network structure and hyper-parameters can be found in the Appendix.

3.2 Offline RL Training with Single-agent Dataset

In this part, we will introduce how we collect the single-agent dataset and how to utilize this dataset to learn multi-agent control models.

3.2.1 Single-agent Dataset Collection

To collect an expert single-agent dataset, we first obtain a single-agent AI, denoted as WeKick, from self-play league training. WeKick took first place at the Google Research Football Competition 2020 (Google, 2020) and it is the most powerful football AI in the world until now. We let WeKick play with itself and store all the battle data, including raw observations, actions, and rewards. During the self-playing, only the designated player at one side can be controlled by the WeKick, and the designated player is not fixed and is changed automatically according to the build-in strategy. The game will last for $3,000$ steps for each round (or episode). At last, we collected $21,947$ episodes from the self-play. This dataset will then be used for training our multi-agent offline RL model and other offline RL baselines.

3.2.2 Multi-agent Offline RL Training

The dataset is collected from single-agent control, so it is easy to train a single-agent model with behavior cloning. However, such trained models can not be applied to multi-agent control and many problems will be raised. For example, we find that if we control all the ten players on the court with the same trained model, all the players will be huddled together because all the players tend to get the ball. In single-agent playing, we only need to control the player who is closest to the ball most time. And the designated player is dynamically changed which means the observation inputs for the model are switched between different players. However, for the multi-agent control, we need to control the player who is far from the ball and the observation inputs for each control model are always from a specific player. To handle the differentiation between single-agent and multi-agent playing, we carefully designed the observations, actions, and our learning algorithm.

As for the observations, we construct a 268-dimensional vector (as described in Section 3.1) from the raw observation for each player. As for the actions, we extend the original 19 discrete actions to 20 discrete actions by adding an extra build-in action. All the non-designated players will be assigned the build-in action. When the player takes the build-in action, the player will behave like a build-in agent. Currently, such build-in agents are obtained from a rule-based tactic, and we will try to convert it to a learning-based controller in future work. To be clear, the build-in action is only used for the full game and dropped for all the academic scenarios.

The most direct way to build a multi-agent AI is to train behavior-cloning models with the pre-processed data. The observations-state pair is represented as $\{(o^i_{1:t}, u^i_t)\}^n_{i=1}$, where $o^i_{1:t}$ is the observation history of agent i, u^i_t is the action of agent i at time-step t and n is the agent number ($n = 10$ in the GRF full game). Our goal is to train a parametric neural network policy $\pi_\theta(u^i|o^i_{1:t})$ to mimic the strategy implied in the data, where θ is the parameter of the policy network and all the

agents share the same parameter. The policy will be updated via behavior cloning with log loss:

$$\mathcal{L}_{\text{bc}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\frac{1}{T}\sum_{t=1}^{T}-\frac{1}{n}\sum_{i=1}^{n}\log\pi_\theta(u_t^i|o_{1:t}^i)\right], \tag{1}$$

where $\tau = \{o_t^i, u_t^i, r_t\}_{t=1:T}^{i=1:n}$ is the joint trajectory sampled from the dataset $\mathcal{D}$.In the experiment, we find the agent directly trained with behavior cloning tends to only choose the build-in action. This is because there is a serious class imbalance problem in the dataset, i.e., the number of build-in actions is much larger than other actions. To relieve the class imbalance problem, we introduce a weighting factor into the log loss. Similar to focal loss (Lin et al., 2017), we use an α-balanced variant of the log loss:

$$\mathcal{L}_{\alpha-\text{balance}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\frac{1}{T}\sum_{t=1}^{T}-\frac{1}{n}\sum_{i=1}^{n}\alpha(u^i)\log\pi_\theta(u_t^i|o_{1:t}^i)\right], \tag{2}$$

where $\alpha(u^i)$ is the weighting factor function, which is conditioned on the action label u^i. Besides, there is a designated player among the ten players who should not use the build-in action, so we add the extra loss to minimize the log probability of the build-in action of the designated player:

$$\mathcal{L}_{\text{min\_build\_in}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{n}-I(i,t)\log\left(1-\pi_\theta(u_{\text{build\_in}}^i|o_{1:t}^i)\right)\right], \tag{3}$$

where $u_{\text{build\_in}}^i$ is represented for the build-in action and $I(i,t)$ is an indicator function, which is defined as:

$$I(i,t) = \begin{cases} 0 & \text{if agent } i \text{ is a non-designated player at time step } t \\ 1 & \text{if agent } i \text{ is a designated player at time step } t. \end{cases} \tag{4}$$

Using past good experiences as the supervision has shown great success in many control tasks (Oh et al., 2018; Vinyals et al., 2019; Zha et al., 2021). Similar to the ranking buffer trick as proposed by Zha et al. (2021), we weight each trajectory in the dataset via their cumulative rewards. And trajectories with higher cumulative rewards receive higher weight and contribute more in the training loss:

$$\mathcal{L}_{\text{buffer\_ranking}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\rho(\tau)\frac{1}{T}\sum_{t=1}^{T}-\frac{1}{n}\sum_{i=1}^{n}\alpha(u^i)\log\pi_\theta(u_t^i|o_{1:t}^i)\right]. \tag{5}$$

where $\rho(\tau)$ is the trajectory weighting factor. And the practical setting of the trajectory weighting factor can be found in the Appendix.

To drive a policy improvement, we further adopt the Advantage-Weighted Regression (AWR) (Peng et al., 2019). We first train a centralized value network $V_\phi^{\mathcal{D}}(s_{1:t})$ using the rewards from the dataset. And then compute the advantage with cumulative rewards (or reward return) $\mathcal{R}^{\mathcal{D}}$:

$$A(\mathbf{u}_t, s_{1:t}) = \mathcal{R}^{\mathcal{D}} - V_\phi^{\mathcal{D}}(s_{1:t}). \tag{6}$$

Then the advantage weight is defined as:

$$w(\mathbf{u}_t, s_{1:t}) = \text{clip}\left(\exp\left(\frac{1}{\beta}A(\mathbf{u}_t, s_{1:t})\right), w_{\min}, w_{\max}\right), \tag{7}$$

where $\beta > 0$ is a fixed temperature parameter and clip$(\cdot)$ and we utilize the clipping function to limit the weight to the range of $w_{\min}$ to $w_{\max}$. The clipping function is helpful to relieve gradient explosion and gradient vanishing problems. The advantage weight will be applied to the training loss:

$$\mathcal{L}_{\text{adv}} = \mathbb{E}_{\tau\sim\mathcal{D}}\left[\rho(\tau)\frac{1}{T}\sum_{t=1}^{T}-\frac{w(\mathbf{u}_t, s_{1:t})}{n}\sum_{i=1}^{n}\alpha(u^i)\log\pi_\theta(u_t^i|o_{1:t}^i)\right], \tag{8}$$

In conclusion, the final training loss of our offline algorithm is the combination of the advantage-weighted loss and build-in action minimize loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{adv}} + \eta\mathcal{L}_{\text{min\_build\_in}}, \tag{9}$$

where η is a loss balancing coefficient (i.e., a fixed hyper-parameter) and the loss is optimized with the Adam optimizer.

	CQL	MABC	+α	+MinBuild-in	+BR	+AW
Win Rate	0	0.834	0.890	0.926	0.930	**0.944**
Draw Rate	0.258	0.112	0.088	0.064	0.058	**0.036**
Failure Rate	0.742	0.054	0.022	**0.010**	0.012	0.02
Goal Diff	-1.496	2.140	2.732	3.056	2.816	**3.096**
TrueSkill	6.33 ± 3.16	19.33 ± 3.21	20.88 ± 3.20	22.18 ± 3.23	22.39 ± 3.23	$\mathbf{22.84} \pm 3.25$

Table 1: Evaluation results of different algorithms on the multi-agent GRF full game. Results show that our final algorithm (+AW) achieves the best performance with the highest rate of winning and the highest goal difference. The results of the TrueSkill evaluation are shown in the last row and mean scores with the standard deviation are reported. Our final TiKick model achieves the highest TrueSkill score.

4 Experiments

In this section, we will evaluate our algorithm on the GRF full game and also show how the pre-trained model can accelerate multi-agent reinforcement learning in academic scenarios.

4.1 Multi-Agent Google Research Football Full Game Evaluation

In the multi-agent GRF full game, we need to control all the players except the goalkeeper. To the best of our knowledge, Tikick is the first learning-based AI system that can take over the multi-agent GRF full game, while previous work could either control a single agent or experiment on toy academic scenarios. To evaluate our proposed offline reinforcement learning algorithm, we construct a series of incremental baselines from the vanilla multi-agent behavior cloning (MABC) algorithm.

Baselines:

CQL: CQL (Kumar et al., 2020) is an offline reinforcement learning algorithm that tries to learn conservative Q-values via adding penalties on the Q-functions.

MABC: The multi-agent behavior cloning algorithm has been described in Section 3.2.2 and uses a naive supervised loss as shown in Equation 1.

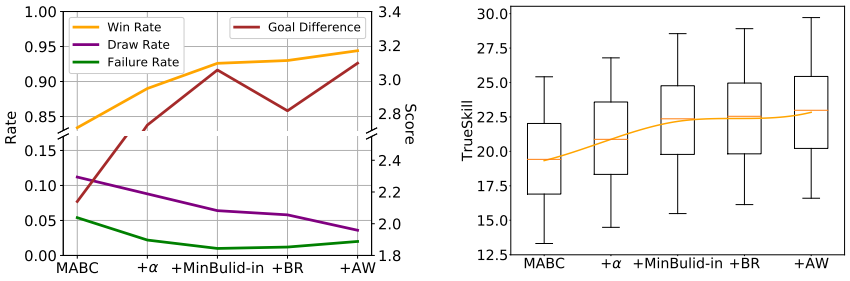
+ α-Balance (abbreviated as +α): Add an α-balance weight to the MABC baseline to relieve the class imbalance problem as shown in Equation 2.

+ Min Build-in (abbreviated as +MinBuild-in): Add a build-in action minimization loss to the previous baseline to force the designated player to choose the non-build-in action as shown in Equation 3.

+ Buffer Ranking (abbreviated as +BR): Add the buffer ranking trick to take advantage of past good experiences as shown in Equation 5.

+ Advantage Weight (abbreviated as +AW): Add an advantage-weighted loss to the previous baseline and obtain the final loss as shown in Equation 9. And this method has been served as the final TiKick model.

All the baselines share the same set of hyper-parameters and the same neural network backbone. Table 1 shows the evaluation results of different algorithms on the multi-agent GRF full game. We report the rate of winning, draw, and failure, and also the goal differences after combating with the build-in AI for 500 rounds (with $3,000$ steps per round). Results show that CQL fails to defeat the build-in AI with a zero winning rate and our final algorithm (i.e., **+ Advantage Weight**) achieves the best performance with the highest rate of winning and the highest goal difference. Figure 1(a) shows the curves of the changes in the rate of winning, draw, and failure of different algorithms, and the curves of the changes on the goal differences of different algorithms. To further evaluate the performance of different algorithms, we rank all the algorithms with the TrueSkill rating system (Herbrich, Minka, and Graepel, 2006), and the results are shown in the last row of Table 1 and Figure 1(b). And our final TiKick model achieves the highest TrueSkill score.



(a) (b)

Figure 1: (a) The curves of the changes on the rate of winning, draw, and failure, and the goal differences of different algorithms. And the curve of the changes on the goal differences of different algorithms. Our final algorithm (+AW) achieves the best performance with the highest rate of winning and the highest goal difference. (b) TrueSkill evaluation for different algorithms. And our final TiKick model achieves the highest TrueSkill score.

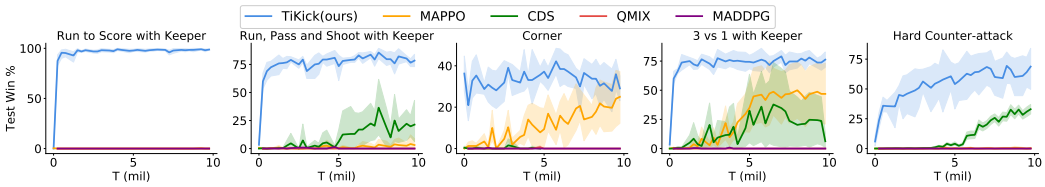


Figure 2: Comparison of our method against baseline algorithms on GRF academic scenarios. Results show that our method, denoted as TiKick, achieves the best performance and the lowest sample complexity on all the scenarios.

4.2 Accelerate Multi-agent Reinforcement Learning Training

Preivous work (Silver et al., 2016; Vinyals et al., 2019; Ye et al., 2020) shows that pre-trained supervised models help a lot for the follow-up deep reinforcement learning training. In this section, we will examine whether our pre-trained offline RL models can accelerate the multi-agent reinforcement learning training process. Thus, we consider five GRF academic scenarios originally proposed in Kurach et al. (2019), including:

- **Run_to_Score_with_Keeper**
- **Run_Pass_and_Shoot_with_Keeper**
- **Corner**
- **3_vs_1_with_Keeper**
- **Hard_Counter-attack**

In this experiment, we use QMIX (Rashid et al., 2018), MADDPG (Lowe et al., 2017), CDS (Li et al., 2021) and MAPPO (Yu et al., 2021) as the baselines.

Baselines:

QMIX: A value-based method that learns a monotonic approximation for the joint action-value function. QMIX factors the joint-action into a monotonic nonlinear combination of individual utilities of each agent. A mixer network with nonnegative weights is responsible for combining the agent's utilities.

MADDPG: A policy-based multi-agent algorithm, which adapts the single-agent DDPG algorithm to the multi-agent setting.

CDS: A value-based multi-agent algorithm with mutual information maximization between agents' identities and their trajectories. To be clear, CDS only combats with three enemies in the

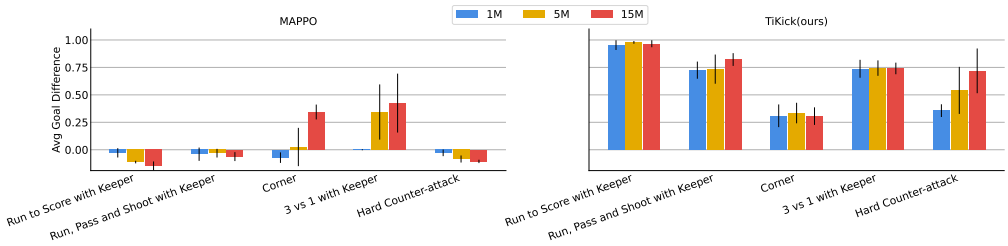


Figure 3: The average goal difference on all the academic scenarios for MAPPO (left) and TiKick (right). Results show that our method achieves the highest scores on all the scenarios and our method can reach the highest scores within just 1M steps on four of five scenarios.

Hard_Counter-attack scenario, while our method can combat with eleven enemies, which is much harder than the four-enemy version.

MAPPO: A policy-based method that uses a centralized value function without individual utilities. MAPPO adapts the single-agent PPO algorithm to the multi-agent setting. Many useful tricks are used to stabilize the training, including Generalized Advantage Estimation (GAE) (Schulman et al., 2015), PopArt (Hessel et al., 2019), observation normalization, value clipping, layer normalization, and ReLU activation with orthogonal initialization.

Our method is built on the MAPPO algorithm and we load weights from our pre-trained TiKick offline RL model. Because there is a minor difference between the GRF full game and academic scenarios (i.e., academic scenarios have no build-in action), we do not load the weights of the last layer in the pre-trained model. In the experiment, we measure the winning rates and goal differences against the build-in AI over 5 seeds for all the algorithms and each evaluation data point is obtained from 64 test rounds.

We show the training performance comparison against baselines in Figure 2. Results show that our method, denoted as TiKick, achieves the best performance and the lowest sample complexity on all the scenarios. Figure 3 shows the average goal difference on all the academic scenarios for MAPPO and TiKick. Results show that our method achieves the highest scores on all the scenarios and our method can reach the highest score within just 1M steps on four of five scenarios. All the experiments show that the pre-trained offline RL model can accelerate the multi-agent RL training process.

5 Conclusion

In this paper, we have proposed a new multi-agent offline reinforcement learning method for the Google Research Football full game. Our proposed learning framework, denoted as TiKick, is implemented practically by using deep recurrent neural networks and carefully designed training losses. Experimental results show that our final algorithm achieves the best performance on the GRF full game. Furthermore, experiments on the GRF academic scenarios show that our pre-trained model can accelerate the multi-agent RL training process with better performance and lower sample complexity. In the future, we will try to apply our pre-trained model to the multi-agent league training to further improve the performance and robustness.

References

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Google. 2020. Google research football competition 2020. `https://www.kaggle.com/c/google-football`.

Herbrich, R.; Minka, T.; and Graepel, T. 2006. Trueskill™: A bayesian skill rating system. In *Proceedings of the 19th international conference on neural information processing systems*, 569–576.

Hessel, M.; Soyer, H.; Espeholt, L.; Czarnecki, W.; Schmitt, S.; and van Hasselt, H. 2019. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3796–3803.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1997. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, 340–347.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*.

Kurach, K.; Raichuk, A.; Stanczyk, P.; Zajkac, M.; Bachem, O.; Espeholt, L.; Riquelme, C.; Vincent, D.; Michalski, M.; Bousquet, O.; et al. 2019. Google research football: A novel reinforcement learning environment. *arXiv preprint arXiv:1907.11180*.

Li, C.; Wu, C.; Wang, T.; Yang, J.; Zhao, Q.; and Zhang, C. 2021. Celebrating diversity in shared multi-agent reinforcement learning. *arXiv preprint arXiv:2106.02195*.

Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; and Dollár, P. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, 2980–2988.

Liu, S.; Lever, G.; Merel, J.; Tunyasuvunakool, S.; Heess, N.; and Graepel, T. 2019. Emergent coordination through competition. *arXiv preprint arXiv:1902.07151*.

Liu, S.; Lever, G.; Wang, Z.; Merel, J.; Eslami, S.; Hennes, D.; Czarnecki, W. M.; Tassa, Y.; Omidshafiei, S.; Abdolmaleki, A.; et al. 2021a. From motor control to team play in simulated humanoid football. *arXiv preprint arXiv:2105.12196*.

Liu, X.; Jia, H.; Wen, Y.; Yang, Y.; Hu, Y.; Chen, Y.; Fan, C.; and Hu, Z. 2021b. Unifying behavioral and response diversity for open-ended learning in zero-sum games. *arXiv preprint arXiv:2106.04958*.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

Mahajan, A.; Rashid, T.; Samvelyan, M.; and Whiteson, S. 2019. Maven: Multi-agent variational exploration. *arXiv preprint arXiv:1910.07483*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Oh, J.; Guo, Y.; Singh, S.; and Lee, H. 2018. Self-imitation learning. In *International Conference on Machine Learning*, 3878–3887. PMLR.

Peng, X. B.; Kumar, A.; Zhang, G.; and Levine, S. 2019. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, 4295–4304. PMLR.

Saxe, A. M.; McClelland, J. L.; and Ganguli, S. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Taiga, A. A.; Fedus, W.; Machado, M. C.; Courville, A.; and Bellemare, M. G. 2019. On bonus based exploration methods in the arcade learning environment. In *International Conference on Learning Representations*.

Team, E. L.; Stooke, A.; Mahajan, A.; Barros, C.; Deck, C.; Bauer, J.; Sygnowski, J.; Trebacz, M.; Jaderberg, M.; Mathieu, M.; et al. 2021. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575(7782):350–354.

Ye, D.; Chen, G.; Zhao, P.; Qiu, F.; Yuan, B.; Zhang, W.; Chen, S.; Sun, M.; Li, X.; Li, S.; et al. 2020. Supervised learning achieves human-level performance in moba games: A case study of honor of kings. *IEEE Transactions on Neural Networks and Learning Systems*.

Yu, C.; Velu, A.; Vinitsky, E.; Wang, Y.; Bayen, A.; and Wu, Y. 2021. The surprising effectiveness of mappo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*.

Zha, D.; Lai, K.-H.; Zhou, K.; and Hu, X. 2021. Simplifying deep reinforcement learning via self-supervision. *arXiv preprint arXiv:2106.05526*.

Zhang, T.; Xu, H.; Wang, X.; Wu, Y.; Keutzer, K.; Gonzalez, J. E.; and Tian, Y. 2020. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*.

Ziyang Li, Kaiwen Zhu, F. Z. 2020. Wekick. `https://www.kaggle.com/c/google-football/discussion/202232`.

A Trajectory Weighting Factor Design

In this section, we will describe how we design the trajectory weighting factor. Given a joint trajectory $\tau = \{o_t^i, u_t^i, r_t\}_{t=1:T}^{i=1:n}$ sampled from the dataset, we can calculate the cumulative reward as:

$$\mathcal{R}(\tau) = \sum_{t=1}^{T} r_t. \tag{10}$$

Then we define a simple trajectory weighting factor as:

$$\rho(\tau) = \begin{cases} \rho_0 & \text{if } \mathcal{R}(\tau) < \mathcal{R}_{\text{threshold}} \\ \rho_1 & \text{if } \mathcal{R}(\tau) \geq \mathcal{R}_{\text{threshold}}, \end{cases} \tag{11}$$

where ρ_0, ρ_1 and $\mathcal{R}_{\text{threshold}}$ are hyper-parameters. In this paper, we set $\rho_0 = 0, \rho_1 = 1$ and $\mathcal{R}_{\text{threshold}} = 3$.

B Hyper-parameters

In this section, we will describe the hyper-parameters in detail.

Hyper-parameters	Value
policy network learning rate	$1e-4$
critic network learning rate	$1e-4$
number of environment parallel threads	64
number of MLP layers	4
number of GRU layers	1
training episode length	1,500
test episode length	3,000
data chunk length	25
observation size	268
action size	20 for the GRF full game, 19 for academic scenarios
number of evaluation round	500 for the GRF full game, 64 for academic scenarios

C Observation Design

In this section, we will describe our observation design in detail. Our observation consists of the following parts:

absolute coordinates of players at our side
moving directions of players at our side
absolute coordinates of enemies
moving directions of enemies
absolute coordinate of current player
absolute coordinate of the ball
moving direction of the ball
one-hot indicator for the ball-owned team
one-hot indicator for the active player
one-hot indicator for the game mode
sticky actions
distance between the ball and current player
tired factors of players of our side
yellow card indicator of our side
red card indicator of our side
offside indicator of our side
offside indicator of enemies
distance between the enemies and current player
relative coordinates of players at our side
relative coordinates of players at our side
relative coordinates of players at our side
number of steps left till the end of the match
number of steps left till the end of the half-time
number goal differences
zero vector of 27 dimensions (used for future usage)