Journal of Machine Learning Research 24 (2023) 1-12 Submitted 2/22; Revised 1/23; Published 3/23

MALib: A Parallel Framework for Population-based Multi-agent Reinforcement Learning

Ming Zhou[1,†] MINGAK@SJTU.EDU.CN
Ziyu Wan[1] ALEX_WAN@SJTU.EDU.CN
Hanjing Wang[1] WANGHANJINGWHJ@SJTU.EDU.CN
Muning Wen[1] MUNING.WEN@OUTLOOK.COM
Runzhe Wu[1] RUNZHE@SJTU.EDU.CN
Ying Wen[1,†] YING.WEN@SJTU.EDU.CN
Yaodong Yang[2] YAODONG.YANG@PKU.EDU.CN
Yong Yu[1] YYU@APEX.SJTU.EDU.CN
Jun Wang[3] JUN.WANG@CS.UCL.AC.UK
Weinan Zhang[1] WNZHANG@SJTU.COM

[1] *Department of Computer Science and Engineering, Shanghai Jiao Tong University*
[2] *Institute for Artificial Intelligence, Peking University*
[3] *Department of Computer Science, University College London*
[†] *corresponding authors*

Editor: Joaquin Vanschoren

Abstract

Population-based multi-agent reinforcement learning (PB-MARL) encompasses a range of methods that merge dynamic population selection with multi-agent reinforcement learning algorithms (MARL). While PB-MARL has demonstrated notable achievements in complex multi-agent tasks, its sequential execution is plagued by low computational efficiency due to the diversity in computing patterns and policy combinations. We propose a solution involving a stateless central task dispatcher and stateful workers to handle PB-MARL's subroutines, thereby capitalizing on parallelism across various components for efficient problem-solving. In line with this approach, we introduce MALib, a parallel framework that incorporates a task control model, independent data servers, and an abstraction of MARL training paradigms. The framework has undergone extensive testing and is available under the MIT license (`https://github.com/sjtu-marl/malib`).

Keywords: Multi-agent Learning, Software, Open-Source, Ray, Python

1. Introduction

Population-based multi-agent reinforcement learning has shown great potential in nontrivial multi-agent tasks (Berner et al., 2019; Vinyals et al., 2019; McAleer et al., 2020; Jaderberg et al., 2017) by coordinating dynamical population selection and MARL algorithms (Heinrich and Silver, 2016; Salimans et al., 2017). However, the training of PB-MARL methods is computationally expensive due to the increasing size of policy pools and interactions among them. A feasible way to handle such a computationally expensive task is distributed

computing, especially for reinforcement learning (RL) tasks with irregular training workloads including optimization, rollout, planning (Liang et al., 2018) , etc.Despite the existing work giving sophisticated distributed solutions for RL (Espeholt et al., 2018; Petrenko et al., 2020), they primarily focus on single-agent cases and offer less consideration of multi-agent. Furthermore, PB-MARL involves bi-level heterogeneity due to inner-loop MARL policy optimization and outer-loop policy combinations generation. This bi-level heterogeneity (see Figure 1) makes PB-MARL's parallelism more complex than existing distributed MARL frameworks, which typically focus on fixed policy combinations (Liang et al., 2018; Pretorius et al., 2021). Specifically, because the outer-loop (see Figure 1(a)) involves a growing policy population and flexible agent selection, the policy combinations can be heterogeneous and difficult to solve. Additionally, because the inner-loop of PB-MARL involves executing MARL tasks, it also inherits the computing heterogeneity of RL, which adds another layer of complexity to the overall framework.

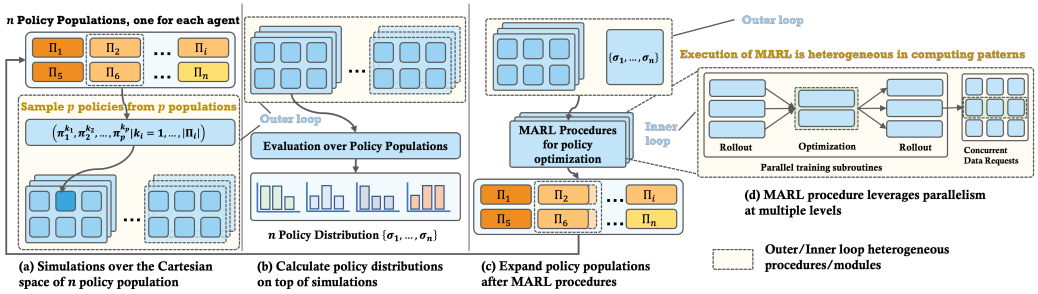


Figure 1: A PB-MARL algorithm starts from n policy populations and runs simulations for policy combinations (a), followed by evaluations to compute policy distribution for each of populations (b). Parallel MARL procedures learn new policies by playing against policies sampled from these populations (d), and then these new policies will be used to expand populations (c). The algorithm executes (a) $\sim$ (d) in a loop until convergence.

As a solution to these challenges, we present a parallel PB-MARL framework, *MALib*, which considers the ability to handle bi-level heterogeneity in PB-MARL and the algorithm abstraction. For the former, we decompose the controlling logic into different layers according to the dependency and functionality, which are correspondingly responsible for parallel control and policy interaction control. For the latter, we give the algorithm abstractions from the perspective of multi-agent optimization paradigms. This provides full customization for low-level training logic and reserved customization for mid-level interaction logic with predefined templates, trading off between performance and ease of use.

2. Related Work

PB-MARL aims to address complex multi-agent learning tasks by simplifying the joint policy space, incorporating large-scale simulations, it combines population-based training and metagame analysis (Wellman, 2006; Jaderberg et al., 2017; Carroll et al., 2019; Jaderberg et al., 2019; Lanctot et al., 2017; Muller et al., 2020; Nieves et al., 2021; Vinyals et al., 2019; Heinrich et al., 2015). One difficulty in PB-MARL is the quadratically increasing and heterogeneous policy interactions, which results in expensive computing costs. As learned

from existing research for improving the efficiency of large-scale tasks, distributed computing is shown to be a promising solution. There are many distributed frameworks designed for deep RL (Flajolet et al., 2022; Nair et al., 2015; Mnih et al., 2016; Stooke and Abbeel, 2018; Babaeizadeh et al., 2017), but most of them focus on single-agent cases. Recently some frameworks (Liang et al., 2018; Pretorius et al., 2021; Hu et al., 2022) have exploited parallelism in accelerating deep MARL algorithms. However, their architectures and abstractions do not offer efficient yet user-friendly interfaces to those population-based methods, making them poorly fit for the research of PB-MARL. As for existing distributed frameworks for population-based learning (Flajolet et al., 2022; Zhi et al., 2020), they have limited to single-agent cases and lack specialist designs that address heterogeneous policy interactions.

3. Framework Description

The development of *MALib* is based on Python, Ray (Moritz et al., 2018) and PyTorch (Paszke et al., 2019). Conceptually, *MALib* packs up hierarchical abstractions for PB-MARL pipelines, where the essentials include: (1) a control model implementing a *semipassive* mechanism to handle bi-level heterogeneity, (2) independent data servers supporting high data parallelism and (3) unified abstractions for population selection and optimization paradigms. The following content introduces the critical components of *MALib*, also the philosophy to address the bi-level heterogeneity. *MALib* opens up broad customization interfaces so that users can easily customize the algorithms, environments, etc.

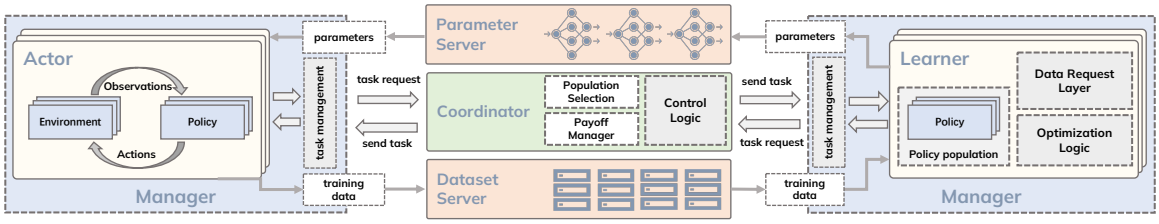


Figure 2: The Coordinator dispatches tasks to workers (*Actors* and *Learners*), and the workers coordinate with the Coordinator in semipassive, i.e., workers send task requests to the Coordinator instead of waiting. *Actors* for simulation and data collection, *Learners* for policy optimization. *Managers* are designed for worker coordination. All workers work in parallel and exchange data via data servers.

Control Model & Task Dispatching. It is common to use decentralized or centralized control to handle heterogeneity and task dispatching for many existing distributed RL frameworks (Liang et al., 2018; Espeholt et al., 2018; Petrenko et al., 2020). However, PB-MARL's bi-level heterogeneity, intensive computation, and complex module dependencies result in a high cost for centralized and decentralized control to implement the control logic. We address this problem by introducing a control model that executes semipassively. The semipassive mode is a mix of conventional centralized control and decentralized control, in which the control logic and the corresponding cost for computation and intermediate state maintenance are offloaded from the central node (*Coordinator*), hierarchically split, and assigned to workers (*Actors*, *Learners*), as illustrated in Figure 2. In this mechanism, the

controller kicks in only when the subroutines have finished and further processing is required from workers. Then, detailed task descriptions for information such as policy combination, agents involved and etc., are generated and dispatched to the target workers. With this mechanism, a learning task in *MALib* is decoupled into multiple rollout and optimization tasks, which are delegated to *Actors* and *Learners*, respectively.

Independent Data Servers. We decouple the data flow from a task execution flow so that the workers can execute fully asynchronously, which differs from existing frameworks like MAVA (Pretorius et al., 2021). Specifically, there is a *Parameter Server* and *Dataset Server* for parameters and training data coordination between the workers, respectively. As the training of PB-MARL is highly concurrent, such centralized data servers create an unbalance in data reading and writing, resulting in low data throughput and training efficiency. Therefore, we use independent data pipelines/processes for concurrent tasks and incorporate fine-grained locks for read-write balance, so that the data request can be further executed asynchronously. Moreover, thanks to Ray's Plasma (Robert, 2016), *MALib* can further decrease the influence of data serialization on throughput [1].

Abstractions for Optimization and Population Selection. As the optimization of MARL is miscellaneous, an ideal design is a unified optimization interface for code reuse. We implement an *AgentInterface* that decouples the optimization pipeline as model-coordination, data request, and optimization logic. With the model-coordination, users can easily implement complex MARL optimization, such as centralized training (Rashid et al., 2018), networked learning (Zhang et al., 2018), etc. The optimization can be distributed to multiple parallel instances to perform asynchronous or synchronous gradient updates. Therefore, *MALib* can satisfy different distributed training strategies, which opens up further research on distributed MARL algorithms. We also provide the abstraction for population-selection in *Coordinator*, which is a key to PB-MARL as it determines how to expand the policy population and generate policy interactions.

Environments. *MALib* implements a unified environment interface to integrate typical RL environments, e.g., OpenSpiel (Lanctot et al., 2019), Gym (Brockman et al., 2016), StarCraftII (Samvelyan et al.), Google Research Football (Kurach et al., 2020) and PettingZoo (Terry et al., 2020). *MALib* also considers environment vectorization, which enables batch environment stepping.

4. Summary

MALib is an open-source framework for PB-MARL that aims to address the bi-level heterogeneity by introducing a control model that executes in semipassive. Its independent data servers decouple the data flow from task execution and allow the workers to maintain high parallelism. The abstractions of population-selection and optimization paradigms are designed to prompt research on PB-MARL from an engineering-support perspective. Moreover, *MALib* is attracting community interest in secondary development [2].

1. We list the performance comparison results that show our design outperforms the baseline framework in throughput, see issue #35, also refer to the documentation for more details about the data servers.
2. `https://github.com/Shanghai-Digital-Brain-Laboratory/DB-Football`

Acknowledgments

The SJTU team is supported by the "New Generation of AI 2030" Major Project (2018AAA0100900), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), National Natural Science Foundation of China (62076161, 61632017) and Shanghai Sailing Program (21YF1421900). We also thank Zhicheng Zhang, Hangyu Wang, Ruiwen Zhou, Weizhe Chen, Minghuan Liu, Yunfeng Lin, Xihuai Wang, Derrick Goh, and Linghui Meng for many helpful discussions, suggestions and comments on the project, and Ms. Yi Qu for her help with the design work.

Appendices

This appendix introduces some of the key evaluation results of *MALib*, and more results can be found on our project website (see issue #35). The evaluation focuses on both system and algorithm performance, including the comparison of data throughput, training efficiency, and algorithms' convergence performance. All the experiment results are obtained with one of the following hardware settings:

- *System #1*: a 32-core computing node with dual graphics cards;
- *System #2*: a two-node cluster with each node owning 128-core and a single graphics card. All the GPUs mentioned are of the same model (NVIDIA RTX3090).

Appendix A. Throughput Comparisons

We conduct the throughput evaluation on *MALib* and compare the results with some of the existing SOTA distributed RL frameworks, including RLlib (Liang et al., 2018), Sample-Factory (Petrenko et al., 2020) and SEED RL (Espeholt et al., 2020). Especially, Sample-Factory and SEED RL, which are highly tailored for TPU and GPU instances.

Task Settings. As the environment for throughput comparison, we adopt the multi-agent version of Atari games (MA-Atari) from PettingZoo (Terry et al., 2020), a collection of 2D video games with multiple agents. In our experiments, we use the two-player Pong, a video game with RGB-image frames in $12 \times 12 \times 3$ resolution. We compare *MALib* to other high-throughput frameworks, including IMPALA (Espeholt et al., 2018) for SEED RL and RLlib, APPO (Petrenko et al., 2020) for Sample-Factor and *MALib*. Considering the fairness of comparison, the structure of the tested policy network in each framework is kept the same. Specifically, it is constructed as a ConvNet with three convolutional layers, and two fully-connected heads for the actor and critic. The evaluation of training throughput was conducted in different worker configurations over three minutes of continuous training, considering performance fluctuations caused by environment reset, data concatenation, and other factors like threading lock. For each worker, we fixed the number of environments as 100. The number of workers ranges from 1 to 128 to compare the upper bound and bottleneck in the parallelism performance of different frameworks.

Figure 3 shows the comparison results on System #1. With a unified resource limitation, the maximum concurrency and peak performance that each framework can obtain is closely related to its architecture. In the CPU-only setting, RLlib failed to launch with more than 32 workers, while the threshold for GPU-accelerated RLlib is 8 workers. Despite the extra abstraction layer introduced for tackling PB-MARL problems, both the CPU and GPU versions of *MALib* outperform other frameworks in the MA-Atari environment. Specifically, *MALib* has achieved an average FPS at 25.9K with 128 workers in the CPU-only setting, which is 80% more than that of the second place. And *MALib* achieves FPS at 39.6K (64 workers) and 40.6K (128 workers) with GPU acceleration, which are correspondingly 37.3% and 61.8% more than FPS of Sample-Factory, a framework specially tailored for training conventional RL algorithms on a single GPU node. We notice that Sample-Factory suffers minor performance degradation in both settings with large concurrency, which may result from the potential resource competition.

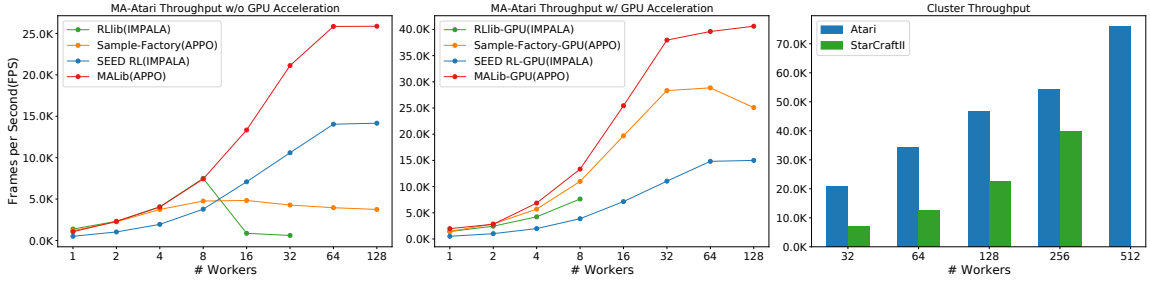


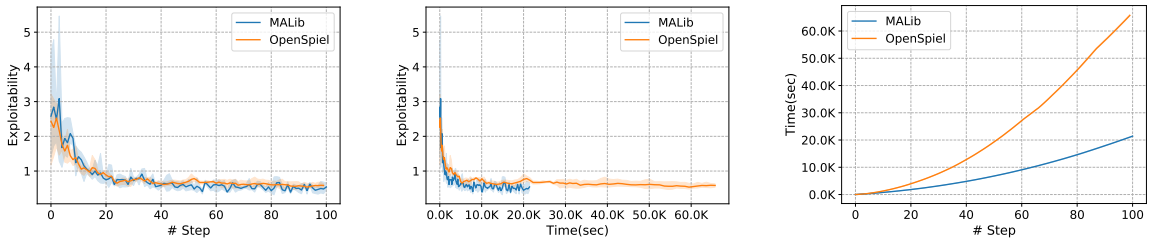
Figure 3: Throughput comparison among the existing RL frameworks and *MALib*. Due to resource limitation (32 cores, 256G RAM), RLlib fails under heavy loads (CPU case: #workers>32, GPU case: #workers>8). *MALib* outperforms other frameworks with only CPU and achieves comparable performance with the highly tailored framework Sample-Factory with GPU despite the higher abstraction introduced. To better illustrate the scalability of *MALib*, we show the MA-Atari and SC2 throughput on System #2 under different worker settings; the 512-workers group on SC2 fails due to resource limitation.

Appendix B. Algorithm Implementation and Performance

A series of algorithms have been implemented in *MALib*, including independent learning algorithms like DQN, PPO, and SAC, along with MARL algorithms like MADDPG and QMIX. Furthermore, these algorithms can be applied to Population-based Training (PBT) and self-play, supported by *MALib*. For the algorithmic performance evaluation, we focus on the convergence rate, which is derived from sample efficiency and training time consumption. The tested algorithms cover both conventional MARL and PB-MARL algorithms. Specifically, we tested PSRO (Muller et al., 2020) for PB-MARL and MADDPG and QMIX for conventional MARL.

Evaluation of PB-MARL. We compared *MALib* with OpenSpiel(Lanctot et al., 2019) on solving Leduc Poker, a common benchmark in Poker AI. To get a relatively accurate

empirical payoff, we run 2,000 simulations for each policy combination, and the maximum of population size is limited to 100.



(a) (b) (c)

Figure 4: Comparisons of *MALib* and OpenSpiel on solving Leduc Poker with PSRO. (a) *MALib* achieves the same performance on exploitability as OpenSpiel; (b) the convergence rate of *MALib* is 3× faster than OpenSpiel; (c) *MALib* achieves a higher execution efficiency than OpenSpiel, since it costs less time consumption to iterate the same learning steps.

We evaluate the convergence of PSRO through the lens of exploitability (McAleer et al., 2020). The lower the exploitability, the closer the algorithm is to an Nash Equilibrium. As shown in Figure 4b, *MALib* cuts 70% execution time while maintaining exploitability with a similar quality as OpenSpiel (Figure 4). Furthermore, we also evaluate other PB-MARL methods like Fictitious Self-Play (FSP) (Heinrich et al., 2015) and Self-Play (SP) (Heinrich and Silver, 2016). Table 1 presents their comparison of execution time and population size when the exploitability decreases to 0.5. The results show that PSRO outperforms the other two methods on both metrics. We argue the reason why SP fails to converge is that Leduc Poker is not a purely transitive game. Thus the SP will be trapped in non-transitivity. As for FSP, it is more time-consuming than PSRO to achieve the same performance. It might be that PSRO considers the interactions and meta-game between different policies in populations and solves it to approximate the Nash Equilibrium of the underlying game, which results in a faster convergence rate. Furthermore, when an exact meta-game solver such as the LP-solver or the α-rank solver (Omidshafiei et al., 2019; Yang et al., 2020) is used, PSRO will converge in a shorter time and with a smaller population size.

Table 1: Comparison of three typical PB-MARL in *MALib*.

Algorithm	**Time(sec)**	**Population Size**
Fictitious Self-Play	7562	60
PSRO(fictitious play)	4862	40
PSRO(α-rank)	**1776**	**21**

Evaluation of Conventional MARL. Multi-agent Particle Environments (MPE) (Lowe et al., 2017) is a typical benchmark environment for the research of MARL. It offers plenty of scenarios covering cooperative and competitive tasks. To investigate the performance of conventional MARL algorithms implemented in *MALib*, we compared with RLlib's imple-

mentation of MADDPG (Lowe et al., 2017). For the evaluation, we considered various task types and varying degrees of sampling parallelism. Specifically, the tasks for the evaluation included cooperative, competitive, and mixed cooperative-competitive tasks; the concurrency of sampling ranges from 8 workers to 128 workers. Figure 5 shows the results on *simple adversary*, a mixed cooperative-competitive task, and it indicates that *MALib*'s implementation performs more steadily than RLlib's. Especially when the worker number increases, RLlib implementation shows high variance and even fails to converge under some settings, while *MALib* does not. We argue the reason is due to the differences in the pipeline scheduling between *MALib* and RLlib. *MALib* executes learning asynchronously, while RLlib, although equipped with parallel sampling, executes sequentially.

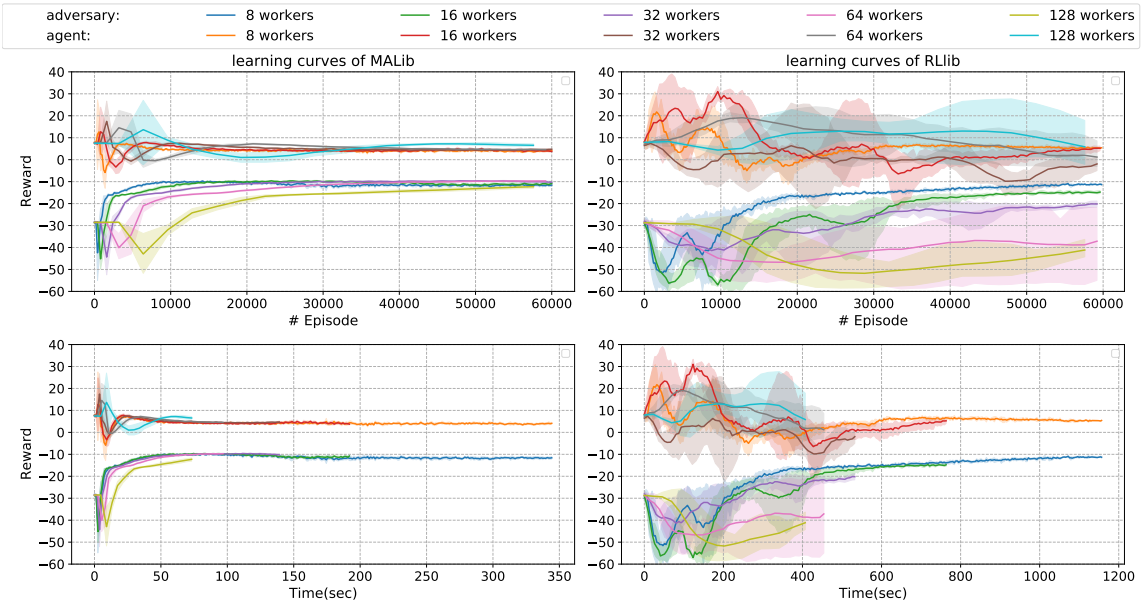


Figure 5: Comparisons of MADDPG in *simple adversary* under different rollout worker settings. Figures in the top row depict each agent's episode reward w.r.t. the number of sampled episodes indicates that *MALib* converges faster than RLlib with equal sampled episodes. Figures in the bottom row show the average time and average episode reward at the same number of sampled episodes, which indicates that *MALib* achieves 5× speedup than RLlib.

References

Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a GPU. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Micah Carroll, Rohin Shah, Mark K. Ho, Tom Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca D. Dragan. On the utility of learning about humans for human-ai coordination. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5175–5186, 2019.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1406–1415. PMLR, 2018.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. Seed rl: Scalable and efficient deep-rl with accelerated central inference. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Arthur Flajolet, Claire Bizon Monroc, Karim Beguir, and Thomas Pierrot. Fast population-based reinforcement learning on a single machine. In *International Conference on Machine Learning*, pages 6533–6547. PMLR, 2022.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 805–813. JMLR.org, 2015.

Siyi Hu, Yifan Zhong, Minquan Gao, Weixun Wang, Hao Dong, Zhihui Li, Xiaodan Liang, Xiaojun Chang, and Yaodong Yang. Marllib: Extending rllib for multi-agent reinforcement learning. *arXiv preprint arXiv:2210.13708*, 2022.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4501–4510, 2020.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, et al. Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E Gonzalez, Michael I Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3059–3068. PMLR, 2018.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

Stephen McAleer, John B. Lanier, Roy Fox, and Pierre Baldi. Pipeline PSRO: A scalable approach for finding approximate nash equilibria in large games. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica.

Ray: A distributed framework for emerging AI applications. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 561–577. USENIX Association, 2018.

Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Pérolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, Zhe Wang, Guy Lever, Nicolas Heess, Thore Graepel, and Rémi Munos. A generalized training approach for multiagent learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

Nicolas Perez Nieves, Yaodong Yang, Oliver Slumbers, David Henry Mguni, Ying Wen, and Jun Wang. Modelling behavioural diversity for learning in open-ended games. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8514–8524. PMLR, 2021.

Shayegan Omidshafiei, Christos Papadimitriou, Georgios Piliouras, Karl Tuyls, Mark Rowland, Jean-Baptiste Lespiau, Wojciech M Czarnecki, Marc Lanctot, Julien Perolat, and Remi Munos. α-rank: Multi-agent evaluation by evolution. *Scientific reports*, 9(1):1–29, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035, 2019.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, pages 7652–7662. PMLR, 2020.

Arnu Pretorius, Kale-ab Tessera, Andries P Smit, Claude Formanek, St John Grimbly, Kevin Eloff, Siphelele Danisa, Lawrence Francis, Jonathan Shock, Herman Kamper, et al. Mava: a research framework for distributed multi-agent reinforcement learning. *arXiv preprint arXiv:2107.01460*, 2021.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 4295–4304. PMLR, 2018.

Nishihara Robert. GitHub - ray-project/plasma: A minimal shared memory object store design — github.com. `https://github.com/ray-project/plasma`, 2016. [Accessed 21-Nov-2022].

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge.

Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, Caroline Horsch, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*, 2020.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 575(7782):350–354, 2019. doi: 10.1038/s41586-019-1724-z.

Michael P Wellman. Methods for empirical game-theoretic analysis. In *AAAI*, pages 1552–1556, 2006.

Yaodong Yang, Rasul Tutunov, Phu Sakulwongtana, and Haitham Bou Ammar. $\alpha\alpha$-rank: Practically scaling α-rank through stochastic optimisation. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1575–1583, 2020.

Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Networked multi-agent reinforcement learning in continuous spaces. In *2018 IEEE conference on decision and control (CDC)*, pages 2771–2776. IEEE, 2018.

Jiale Zhi, Rui Wang, Jeff Clune, and Kenneth O Stanley. Fiber: A platform for efficient development and distributed training for reinforcement learning and population-based methods. *arXiv preprint arXiv:2003.11164*, 2020.