BenchMARL: Benchmarking Multi-Agent Reinforcement Learning

Matteo Bettini[*,1] MB2389@CL.CAM.AC.UK

Amanda Prorok[1] ASP45@CL.CAM.AC.UK

Vincent Moens[2] VMOENS@META.COM

[*] *Work done during an internship at PyTorch, Meta.*

[1] *Department of Computer Science and Technology, University of Cambridge, United Kingdom.*

[2] *PyTorch Team, Meta.*

Abstract

The field of Multi-Agent Reinforcement Learning (MARL) is currently facing a reproducibility crisis. While solutions for standardized reporting have been proposed to address the issue, we still lack a benchmarking tool that enables standardization and reproducibility, while leveraging cutting-edge Reinforcement Learning (RL) implementations. In this paper, we introduce BenchMARL, the first MARL training library created to enable standardized benchmarking across different algorithms, models, and environments. BenchMARL uses TorchRL as its backend, granting it high performance and maintained state-of-the-art implementations while addressing the broad community of MARL PyTorch users. Its design enables systematic configuration and reporting, thus allowing users to create and run complex benchmarks from simple one-line inputs. BenchMARL is open-sourced on GitHub: `https://github.com/facebookresearch/BenchMARL`.

Keywords: Multi-Agent Reinforcement Learning, Benchmark, TorchRL, Vectorization

1 Introduction

The Multi-Agent Reinforcement Learning (MARL) community in PyTorch is evergrowing. Despite this, there exists a persistent fragmentation of tools and standards in the field. The PyTorch-backed TorchRL project (Bou et al., 2023) successfully addressed this issue in the broader Reinforcement Learning (RL) domain, providing a state-of-the-art library adopted by thousands of users. BenchMARL leverages TorchRL's benefits by employing it as the backend in a MARL training library created to enable reproducibility and benchmarking across different MARL algorithms, models, and environments. Its mission is to present a standardized interface that allows easy integration of new algorithms and environments to provide a fair and systematic comparison with existing solutions. Its core design tenets are: (1) Reproducibility, achieved via standardization of configuration; (2) Standardized plotting and reporting, achieved by integrating with the statistically-rigorous tools proposed by Gorsane et al. (2022) [1]; (3) TorchRL backend, which grants high performance and state-of-the-art RL implementations; (4) Experiment independence, achieved via an experiment

1. Based on the NeurIPS 21 Outasnding Paper by Agarwal et al. (2021)

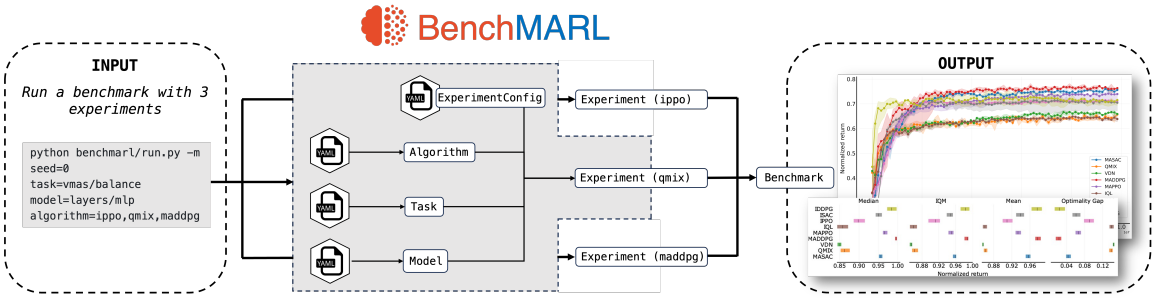


Figure 1: BenchMARL execution diagram. Users run benchmarks as sets of experiments, where each experiment loads its components from the respective YAML configuration files.

class that is agnostic to the choices of algorithm, model, or task; (5) Easy integration of new solutions, achieved via simple abstract interfaces.

Related work. The recent popularity of MARL has exacerbated the fragmentation of shared community standards and tools, with new libraries being frequently introduced, each one focusing on specific algorithms, environments, or models. Popular examples are PyMARL (Samvelyan et al., 2019) and its extensions: PyMARL2 (Hu et al., 2021) and EPyMARL (Papoudakis et al., 2021), which are limited to environments with discrete action spaces. Furthermore, these libraries often implement algorithmic components from scratch, without leveraging native and stable baselines from the single-agent RL community. MARLlib (Hu et al., 2022) addresses this problem by basing on the RLlib framework (Liang et al., 2018). However, RLlib presents significant limitations due to its aim of being agnostic to the underlying learning framework, which causes it to lack core features for state-of-the-art benchmarking, such as support for vectorized environments. This fragmentation of the domain has recently led to a reproducibility crisis, highlighted by Gorsane et al. (2022). While the authors propose a set of tools for results' reporting, there is still the need for a standardized library to run such benchmarks. BenchMARL's mission is to provide such a benchmarking library for MARL, integrating with the reporting tools proposed and using TorchRL as an efficient, tried and tested backend.

2 BenchMARL

BenchMARL tackles its reproducibility goals via defining unifying abstractions over MARL training components. Components are gathered into experiments that are agnostic of their specific implementations. Structured configurations allow to easily run multiple experiments to create a benchmark, making it possible for users to go directly from one-line inputs to benchmarking plots. This process is depicted in Fig. 1. In the following, we illustrate the components and features that enable this pipeline.

2.1 Components

BenchMARL has a few core components, which correspond to classes in the codebase. Each component has a default YAML configuration in a dedicated directory. We now introduce the core components of the library.

Table 1: Algorithms in BenchMARL.

Name	On/Off policy	Actor-critic	Full-observability in critic	Action compatibility	Probabilistic actor
MAPPO (Yu et al., 2021)	On	Yes	Yes	Continuous + Discrete	Yes
IPPO (de Witt et al., 2020)	On	Yes	No	Continuous + Discrete	Yes
MADDPG (Lowe et al., 2017)	Off	Yes	Yes	Continuous	No
IDDPG	Off	Yes	No	Continuous	No
MASAC	Off	Yes	Yes	Continuous + Discrete	Yes
ISAC	Off	Yes	No	Continuous + Discrete	Yes
QMIX (Rashid et al., 2018)	Off	No	NA	Discrete	No
VDN (Sunehag et al., 2017)	Off	No	NA	Discrete	No
IQL (Tan, 1993)	Off	No	NA	Discrete	No

Table 2: Environments in BenchMARL.

Environment	Tasks	Cooperation	Global state	Reward function	Action space	Vectorized
VMAS (Bettini et al., 2022)	5	Cooperative + Competitive	No	Shared + Independent + Global	Continuous + Discrete	Yes
SMACv2 (Ellis et al., 2022)	15	Cooperative	Yes	Global	Discrete	No
MPE (Lowe et al., 2017)	8	Cooperative + Competitive	Yes	Shared + Independent + Global	Continuous + Discrete	No
SISL (Gupta et al., 2017)	2	Cooperative	No	Shared	Continuous	No

Experiment. An experiment is a training run in which an algorithm, a task, and a model are fixed. Experiments are configured by passing these values alongside a seed and the experiment hyperparameters. The experiment hyperparameters cover both on-policy and off-policy algorithms, discrete and continuous actions, and probabilistic and deterministic policies. An experiment can be launched from the command line or from a script.

Benchmark. A benchmark is a collection of experiments that can vary in task, algorithm, or model. A benchmark shares the same hyperparameter configuration across all of its experiments. Benchmarks allow to compare different MARL components in a standardized way. A benchmark can be launched from the command line or from a script.

Algorithms. Algorithms are an ensemble of components (e.g., loss, replay buffer) that determine the training strategy. In Table 1 we report and classify the algorithms available in BenchMARL. We further provide novel implementations (MASAC, ISAC) based on the SAC (Haarnoja et al., 2018) algorithm. All our algorithms have the option of sharing parameters among agent groups for policies and critics. Custom algorithms can be designed by combining this choice with the algorithm and the model. For example, the HetGPPO (Bettini et al., 2023) algorithm can be obtained by using MAPPO without parameter sharing and with a Graph Neural Network (GNN) model for actor and critic.

Tasks. Tasks are scenarios from an environment which constitute the MARL challenge to solve. In Table 2 we report the environments available in BenchMARL. These showcase the variety of MARL paradigms compatible with the library (e.g., differing in cooperation, reward sharing, action space). Importantly, BenchMARL supports vectorized environments, allowing to scale simulation when using batched environments on GPU devices[2].

Models. BenchMARL models are neural network blueprints designed to be used in multiple MARL contexts. They can be instantiated in multiple ways. (1) *Decentralized*: when a model is instantiated in a decentralized manner (e.g., a policy) it will compute a value for each agent. (2) *Centralized with global input*: when a model is instantiated in centralized mode with global input (e.g., a global critic) it will compute a shared value for a group of agents. (3) *Centralized with local input*: like *centralized with global input* with the difference that the value is computed from aggregating local inputs. Furthermore, in any of these

2. For more info, see Bettini et al. (2022).

settings, it is possible to decide whether or not to share parameters among agents. Users can decide which model to use for their critics and actors and are able to mix models as they like. Currently BenchMARL supports Multi Layer Perceptron (MLP) and GNN models[3].

2.2 Features

In the following, we present some of the core features that enable the libraries standardization and reproducibility goals.

Documentation, tests, engineering. The library documentation can be found at this url. We provide an extensive set of examples and notebooks in the dedicated GitHub folder with the goal of showcasing the library's main use-cases. Integration and unit tests are run on all tasks and algorithms. These tests are executed in the continuous integration (CI) and perform complete training iterations to check all components. Coverage is reported on the at this url and is currently around 90%. The library is maintained by the authors and the community with its backend maintained directly by the TorchRL project.

Reporting. The library is directly integrated with the reporting tools proposed by Agarwal et al. (2021) and Gorsane et al. (2022). This allows users to avoid spending time crafting dedicated plots while providing them with state-of-the-art tools. It is furtherly compatible with all the loggers available in TorchRL (e.g., wandb (Biewald, 2020), tensorboard (Abadi et al., 2015), csv) with additional support for saving and restoring experiments.

Configuring. A core reproducibility challenge resides in sharing experiment configurations. To address this BenchMARL uses Hydra (Yadan, 2019), a project that allows to define modular configuration trees in YAML files that can be overridden in many ways either within scripts or in the command line. Such modularity in the configuration allows to run complex benchmarks in one line by listing the desired algorithms, models, and tasks to compare. Different execution backends can be used (e.g., sequential, parallel, slurm).

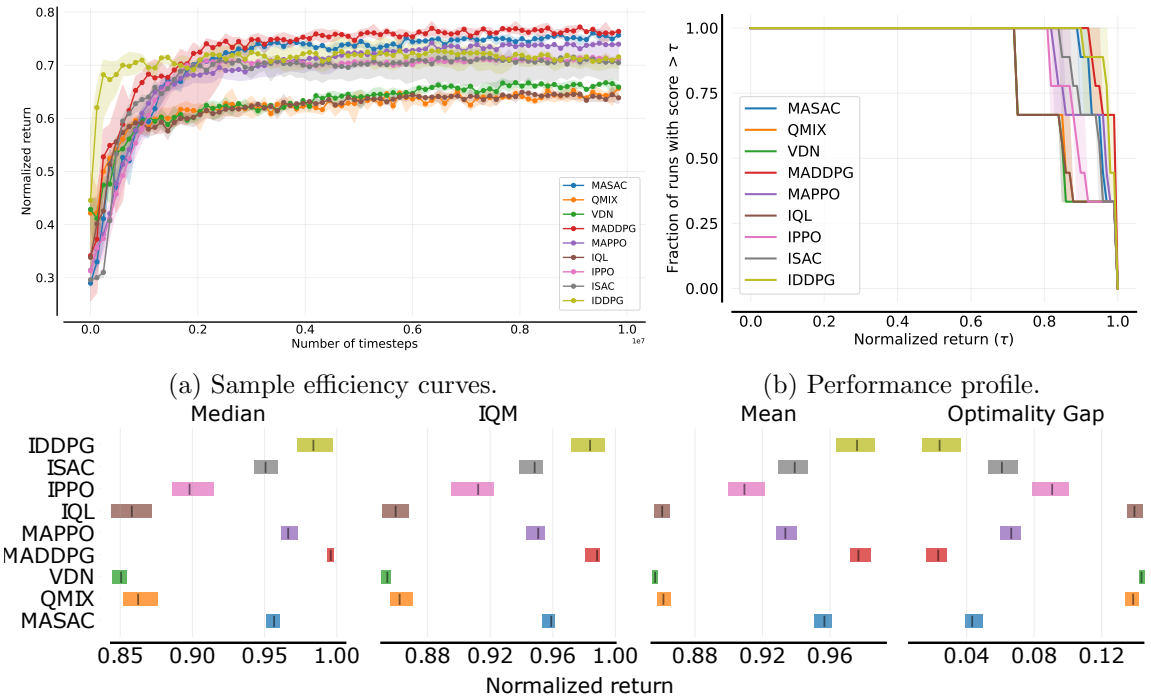
Extending. Each component in the library has an associated abstract class which defines the minimal functionalities needed to implement a new instance. This makes it easy to integrate custom algorithms, models, and tasks allowing to compare them against the wide repository of already implemented ones. Our examples provide detailed illustrations on how to create custom components to enable researchers to benchmark their own solutions.

Public benchmark results. As part of the effort for the standardization of MARL benchmarking, we are fine-tuning and releasing hyperparameters and experiment results for BenchMARL environments in public interactive notebooks . Towards this goal, we have already run and published benchmarks for the VMAS environment (at this url). The results are reported in Appendix A.

3 Conclusion

In this paper we present BenchMARL, the first TorchRL-backed MARL benchmarking library with the goal of enabling standardization and reproducibility. The MARL community can take advantage of BenchMARL to easily compare and share MARL components, increasing reproducibility in the field and reducing its costs. The library also provides an easy-to-use tool for users approaching MARL for the first time.

3. Convolutional Neural Network (CNN) models are being integrated as well.



(a) Sample efficiency curves.

(b) Performance profile.

(c) Aggregate scores.

Figure 2: Benchmark results over VMAS tasks (*Navigation*, *Sampling*, *Balance*). We report the inter-quartile mean (IQM) with 95% stratified bootstrap confidence intervals over 3 random seeds for each experiment (see Gorsane et al. (2022) for more details on the reported metrics). Details and references for the algorithms used are available in Table 1.

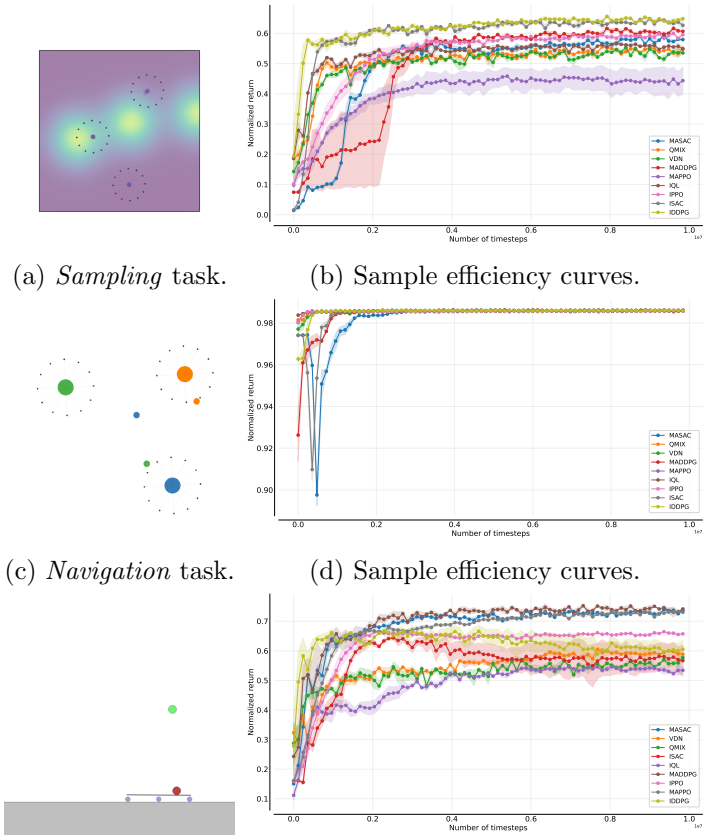
Acknowledgments and Disclosure of Funding

The majority of the work was done during an internship at PyTorch, Meta. This work was supported in part by European Research Council (ERC) Project 949940 (gAIa) and ARL DCIST CRA W911NF-17-2-0181.

Appendix A. Experiment results

In this section, we report the results for the benchmark run on VMAS tasks. For these experiments, we ran all of the currently available algorithms in BenchMARL on the *Navigation*, *Sampling*, and *Balance* tasks. Experiment results, aggregated over all tasks, are reported in Fig. 2. An interactive version of these results is available at `https://wandb.ai/matteobettini/benchmarl-public/reports/VMAS-Benchmarks--Vmlldzo1NzI4MDA5`, where additional metrics can also be accessed. Individual task results are reported in Fig. 3.

All the algorithms, models, and tasks were run using the default BenchMARL configuration `https://github.com/facebookresearch/BenchMARL/tree/main/benchmarl/conf` from v0. The experiment hyperparameters are available in the fine_tuned folder at `https://github.com/facebookresearch/BenchMARL/tree/main/fine_tuned/vmas/conf`.



(a) *Sampling* task. (b) Sample efficiency curves.

(c) *Navigation* task. (d) Sample efficiency curves.

(e) *Balance* task. (f) Sample efficiency curves.

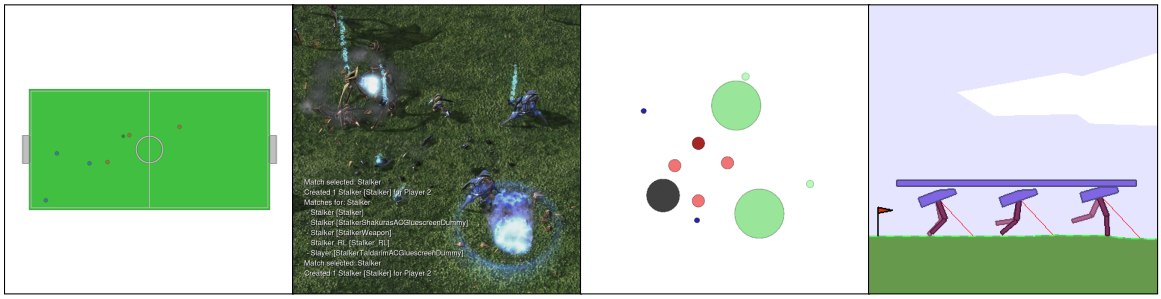
Figure 3: The sample efficiency curves for all BenchMARL algorithms over the three VMAS tasks analyzed. We report the mean with 95% stratified bootstrap confidence intervals over 3 random seeds for each experiment (see Gorsane et al. (2022) for more details on the reported metrics). Details and references for the algorithms used are available in Table 1.

Appendix B. Environments

In Fig. 4 we report the rendering for one example task for each of the environments currently available in BenchMARL. Details and references for all environments are available in Table 2.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin



(a) VMAS (b) SMACv2 (c) MPE (d) SISL

Figure 4: Environments in BenchMARL. This figure shows renderings from one example task for each environment. Details and references for all environments are available in Table 2.

Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. Vmas: A vectorized multi-agent simulator for collective robot learning. In *Proceedings of the 16th International Symposium on Distributed Autonomous Robotic Systems*, DARS '22. Springer, 2022.

Matteo Bettini, Ajay Shankar, and Amanda Prorok. Heterogeneous multi-robot reinforcement learning. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23. International Foundation for Autonomous Agents and Multiagent Systems, 2023.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL `https://www.wandb.com/`. Software available from wandb.com.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch. *arXiv preprint arXiv:2306.00577*, 2023.

Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Benjamin Ellis, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning, 2022. URL `https://arxiv.org/abs/2212.07489`.

Rihab Gorsane, Omayma Mahjoub, Ruan John de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *Advances in Neural Information Processing Systems*, 35:5510–5521, 2022.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International conference on autonomous agents and multiagent systems*, pages 66–83. Springer, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Jian Hu, Siyang Jiang, Seth Austin Harding, Haibin Wu, and Shih wei Liao. Rethinking the implementation tricks and monotonicity constraint in cooperative multi-agent reinforcement learning. 2021.

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

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021. URL `http://arxiv.org/abs/2006.07869`.

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.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

Ming Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. *Machine Learning Proceedings 1993*, pages 330–337, 1993. doi: 10.1016/b978-1-55860-307-3.50049-6.

Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL `https://github.com/facebookresearch/hydra`.

Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.