BenchMARL: Benchmarking Multi-Agent Reinforcement Learning

Matteo Bettini*
University of Cambridge
mb2389@cl.cam.ac.uk

Amanda Prorok
University of Cambridge
asp45@cl.cam.ac.uk

Vincent Moens
PyTorch, Meta
vmoens@meta.com

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 at: https://github.com/facebookresearch/BenchMARL.

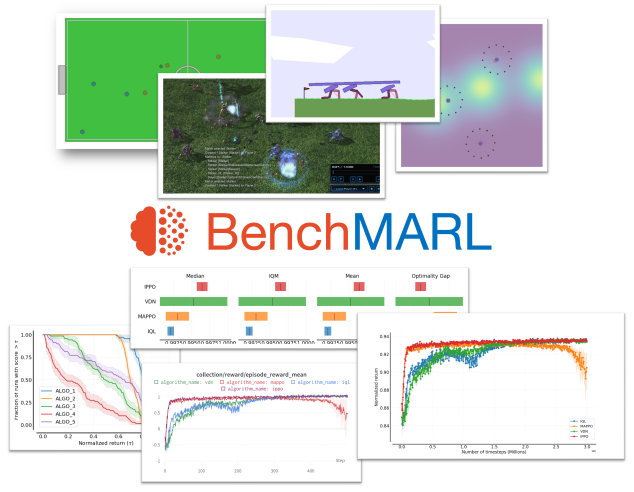


Figure 1: BenchMARL enables comparisons across different Multi-Agent Reinforcement Learning (MARL) algorithms, models, and tasks while focusing on standardization and reproducibility.

1 Introduction

The Multi-Agent Reinforcement Learning (MARL) community in PyTorch is evergrowing. Despite this, there exists a persistent fragmentation of experimental tools and standards in the field. The

*Work done during an internship at PyTorch, Meta.

PyTorch-backed TorchRL project [Bou et al., 2024] successfully addressed this issue in the broader Reinforcement Learning (RL) domain, providing a state-of-the-art library adopted by thousands of users. In this work, we introduce BenchMARL: a MARL training library that leverages TorchRL as its backend while focusing on standardization of MARL experiments.

Thanks to the flexibility of the TorchRL backend, BenchMARL enables 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 using the Hydra framework [Yadan, 2019]; (2) *Standardized plotting and reporting*, achieved by integrating with the statistically-rigorous tools proposed by Gorsane et al. [2022] [2]; (3) *TorchRL backend*, which grants high performance and state-of-the-art RL implementations; (4) *Experiment independence*, achieved via an experiment class that is agnostic to the choices of algorithm, model, or task; (5) *Easy integration of new solutions*, achieved via simple abstract interfaces.

This set of tenets allows BenchMARL users to go from simple command line inputs to statistically rigorous output reports (see Fig. 2) without extensive experimental domain knowledge. In doing so, the library uses (when possible) solutions from the single-agent RL domain, avoiding custom re-implementations and leveraging the extensive benchmarking already performed on TorchRL algorithms [Bou et al., 2024]. High-performance is guaranteed thanks to several features such as: vectorized simulation [Bettini et al., 2022], vectorized training (using `torch.vmap`) over the agents' parameters and data, and a SLURM [Yoo et al., 2003] launcher backend for running experiments on High Performance Computing (HPC) clusters.

In the following, we introduce the library, providing a overview of its structure and discussing its components (Sec. 3.1) and features (Sec. 3.2). We then report results from an initial experimental comparison, consisting of benchmarking the 9 algorithms available in BenchMARL on 3 VMAS [Bettini et al., 2022] tasks representing high-level multi-robot control problems (Sec. 4). With this effort, we hope to stimulate more researchers to adopt shared standards for configuration and reporting towards a standardization of MARL experimental pipelines.

2 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., 2023] 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. In contrast to these PyTorch-based libraries, BenchMARL focuses on covering the whole breadth of MARL algorithms, models, and tasks (e.g., continuous vs discrete actions and state, competitive vs cooperative, vision vs vector observation) while being compatible with state-of the art vectorized simulation.

Concurrent to our work, other MARL libraries have been proposed in the Jax [Bradbury et al., 2018] ecosystem [de Kock et al., 2021, Rutherford et al., 2024]. With Jax being a promising emergent machine learning framework, these projects are important complementary tools to our work,which is in turn focused on the PyTorch domain. To promote cross-framework comparisons, BenchMARL adopts the same reporting format as Mava [de Kock et al., 2021], making interactions easier for experimentalists from either domain. Lastly, we note that these projects often propose single-file implementations, while BenchMARL focuses on providing abstractions and components that can be reused across experiments.

The fragmentation of the MARL 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

[2] Based on the NeurIPS 21 Outstanding Paper by Agarwal et al. [2021]

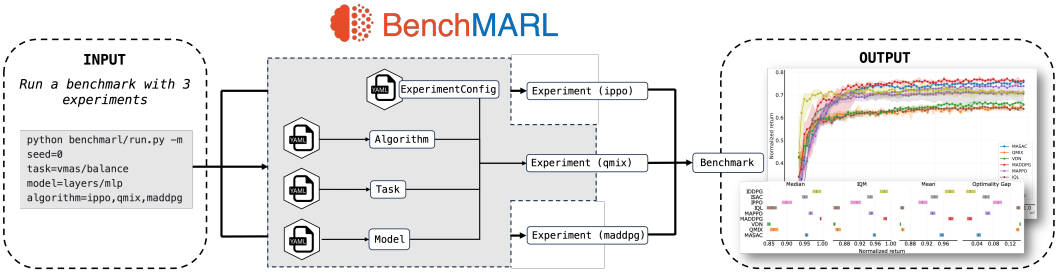


Figure 2: BenchMARL execution diagram. Users run benchmarks as sets of experiments, where each experiment loads its components from the respective YAML configuration files.

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.

3 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. 2. In the following, we illustrate the components and features that enable this pipeline.

3.1 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.

Experiment. An experiment is a training run in which an algorithm, a task, and a model are fixed. Experiments are configured by passing the choices for these components alongside a seed and the experiment's hyperparameters. They cover both on-policy and off-policy algorithms, discrete and continuous actions, probabilistic and deterministic policies, and competitive and cooperative scenarios. Experiments will try to match versions of algorithms and tasks on a best-effort basis: for example, if a task has continuous actions, and an algorithm supports both continuous and discrete actions, the experiment will instantiate the version of the algorithm compatible with the task. To address the heterogeneity of the MARL ecosystem, experiments leverage the agent grouping mechanism available in TorchRL. This mechanism allows to define which agents should have their data stacked in a group (to benefit from vectorization) and which agents should have their data as separate entries (due to heterogeneity in their shapes). This is a unique feature of BenchMARL that allows compatibility with both environments that stack all agent data together (e.g., SMACv2 [Ellis et al., 2022]) and environments that keep agent data separate (e.g., PettingZoo [Terry et al., 2021]). During training, each group will be trained independently and agents within a group will be able to reuse most single-agent RL components, just with the addition of the agent dimension in the data shape. At a high-level, an experiment iteration is composed of the following steps: (1) collection: which is vectorized for compatible environments or parallel/sequential for other environments; (2) addition of data to the replay buffers of each agent group; (3) training: where, for each agent group, data is sampled from the replay buffer and optimizer steps are performed, (4) evaluation: where agents are evaluated on a test environment. An experiment can be launched from the command line using Hydra or directly from a Python script.

Benchmark. A benchmark is a collection of experiments that can vary in task, algorithm, or model. Where possible, a benchmark shares hyperparameters across its experiments. Benchmarks allow to compare different MARL components in a standardized way. A benchmark can be launched from the

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]	27	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
MeltingPot [Leibo et al., 2021]	49	Cooperative + Competitive	Yes	Independent	Discrete	No

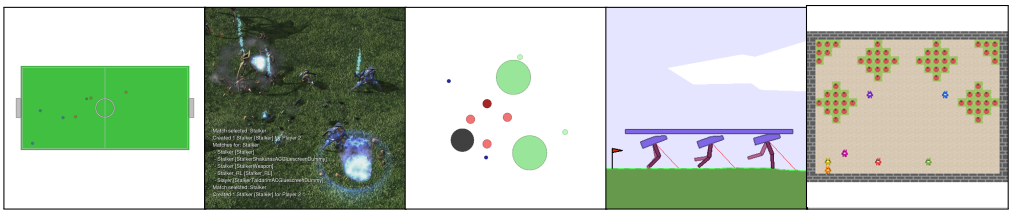
command line using Hydra or directly from a python script. For example, by looking at the input code in Fig. 2, we can see that it is sufficient to provide a list of components (e.g., algorithms, tasks) to instantiate a benchmark. Thanks to Hydra, it is then possible to specify custom backends for running the experiments in a benchmark, such as: sequential, in parallel processes, or in separate SLURM [Yoo et al., 2003] submissions on an HPC cluster.

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. As we can see, BenchMARL provides a wide range of algorithms, allowing our benchmarks to span over the wide range of different options available in the MARL domain. We further provide novel implementations (MASAC, ISAC) based on the SAC [Haarnoja et al., 2018] algorithm. These are multi-agent adaptions of the single-agent algorithm with (MASAC) and without (ISAC) full-observability in the critic. All our algorithms have the option of sharing parameters within agent groups for actors and critics (where applicable). Custom algorithms can further 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 IPPO 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 with the respective number of tasks available in the library. Fig. 3 shows renderings for one example task for each of the environments. Environments in BenchMARL 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[3]. When using a vectorized environment on GPU, both simulation and training can be performed on the same device, granting important performance speed-ups and avoiding data casting and CPU-GPU communication costs. Tasks are represented as enumerations of an environment in order to provide auto-completion functionality when choosing a task and to ground the available tasks associated to a specific BenchMARL release in a static fashion. BenchMARL further supports environments with action masking (e.g., [Ellis et al., 2022]) and allows users to specify custom transforms to process the input-output data of an environment. For example, in environments with visual observations such as MeltingPot [Leibo et al., 2021], transforms are used to store images as integers in the replay buffer, and appropriately cast them to floating point when passing them to neural networks.

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 separate value for each agent. These separate values will only be conditioned on local inputs (e.g., the agent's observation or observations from a local neighborhood). (2) *Centralized with global input*: when a model is instantiated in

[3]For more info, see Bettini et al. [2022].



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

Figure 3: 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.

Table 3: Models in BenchMARL.

Model	Decentralized	Centralized with local inputs	Centralized with global input
MLP	Yes	Yes	Yes
CNN	Yes	Yes	Yes
GRU	Yes	Yes	Yes
LSTM	Yes	Yes	Yes
GNN	Yes	Yes	No
Deep Sets [Zaheer et al., 2017]	Yes	Yes	Yes

centralized mode with global input (e.g., a global critic) it will compute a shared value for a group of agents given a shared input. For example, this could be a centralized critic that takes a global top-down view of the task and outputs a shared value. (3) *Centralized with local input*: like *centralized with global input* with the difference that the value is computed from aggregating local inputs. For example, this could be a centralized critic that takes a concatenation of the agents' observations and outputs a shared value. Furthermore, in any of these 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. The models currently available in BenchMARL are reported in Table 3 and include: Multi Layer Perceptron (MLP), Graph Neural Network (GNN), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Deep Sets [Zaheer et al., 2017] models. BenchMARL models can be chained in a sequence, allowing to build more complex networks. For example, by chaining a CNN to a fully-connected GNN attention layer [Veličković et al., 2018] and an MLP, it is possible to process agents' image inputs using a transformer-like architecture.

3.2 Features

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

Documentation, tests, engineering. The library is accompanied by version-tracked documentation available here. Each component is accompanied by docstrings compiled in the *package reference* section to explain the components' functionalities. We provide an extensive set of examples and notebooks in the dedicated repository folder with the goal of showcasing the library's main use-cases. Integration and unit tests are run on all tasks and algorithms, with a separate job being run for each environment. These tests are executed in the continuous integration (CI) and perform complete training iterations to check all components. Coverage is reported online and is currently around 90%. The library is maintained by the authors and the community with its backend maintained directly in the TorchRL project.

Reporting. The library is directly integrated with the reporting tools proposed by Gorsane et al. [2022]. This allows users to avoid spending time crafting dedicated plots while providing them with state-of-the-art tools for automating this process. To enable this, BenchMARL experiments optionally output results in the JSON format required by the reporting library. Practitioners will be able to use the tools in `https://github.com/instadeepai/marl-eval` to process raw MARL experiment data for downstream use with the tools provided by Agarwal et al. [2021]. BenchMARL 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. Wandb users will be able to access online interactive plots for their experiments together with renderings of evaluation runs.

Configuring. A core reproducibility challenge resides in sharing experiment configurations. For this reason, BenchMARL completely decouples configuration parameters from the Python codebase. Configuration files use a standardized YAML format and are available in the `conf` folder, structured in sub-folders mimicking the hierarchy of the respective Python components. When an experiment is instantiated, each component configuration is loaded from YAML into a corresponding Python dataclass. This process allows to strongly check types and values of a configuration and fail early if some parameters do not pass this check. Loaded BenchMARL components' dataclasses can then be safely passed around as they only represent a lazy configuration and their respective component is not instantiated until the experiment is run. To simplify the process of loading configurations from YAML, users can optionally benefit from using 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). Users that do not want to depend on Hydra can directly load YAML configurations into the respective dataclass from a Python script.

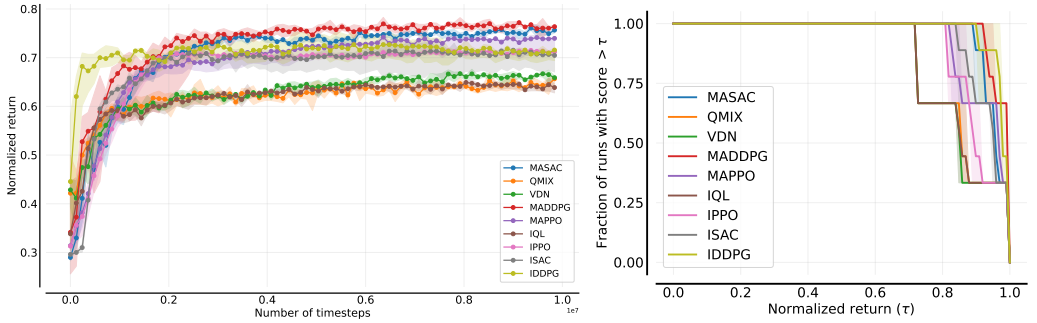
Extending. Each component in the library has an associated abstract class which defines the minimal functionalities needed to implement a new instance. This makes it easy to integrate custom algorithms, models, and tasks allowing to compare them against the wide range of already implemented ones. Our examples provide detailed illustrations on how to create custom components. For example, when creating a new algorithm, users will just need to implement the functions that provide a TorchRL loss and policy as well as some general flags on whether the algorithm should be trained on or off policy and what type of actions are supported. Similarly, when integrating a new environment, they will need to provide a TorchRL environment class alongside basic information on the input and output spaces, global state, agent grouping, and action type. BenchMARL will then automatically be able to determine compatibility between environments and algorithms.

Callbacks and checkpointing. To aid in experimentation, BenchMARL provides multiple callbacks in all the major phases of the training process. These callbacks will notify a list of listener components defined by the user. For example, callbacks can be used: to stop or resume training a particular agent group, to train additional losses on top of the RL objective, to create a curriculum of tasks, to perform self-play in adversarial environments, or simply to log custom data or plots. Alongside callbacks, BenchMARL is compatible with experiment checkpointing and restoring in the standard PyTorch format. Experiment state (e.g., replay buffers, networks, environments) can all be loaded and saved to disk. This particular feature also enables the deployment of trained policies, as shown in Blumenkamp et al. [2024], where policies trained in BenchMARL and VMAS are deployed on a real-world multi-robot system.

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 and algorithms in public interactive notebooks. Fine-tuned configurations are available in the `conf` folder of the project repository. The notebooks and interactive plots are publicly available on Wandb (at this url). Towards this goal, we have already run and published benchmarks for VMAS environments. The results are reported in Sec. 4.

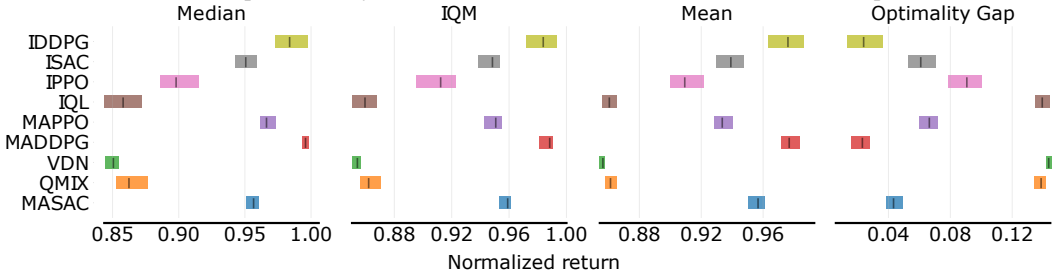
4 Experiments

In this section, we report some experiments to confirm the correctness of the implementations in the library and provide public benchmarking references. Furthermore, since the majority of the algorithms available is simply a multi-agent extension of single-agent algorithms (e.g., PPO, DDPG, SAC, DQN), BenchMARL is able to reuse the algorithm implementations available in TorchRL. We refer to Bou et al. [2024] for extended evaluations of these single-agent algorithms as well as comparisons to the original papers.



(a) Sample efficiency curves.

(b) Performance profile.



(c) Aggregate scores.

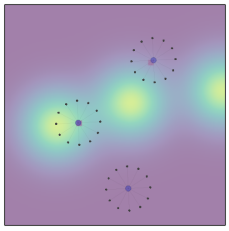
Figure 4: Benchmark results over VMAS tasks (*Navigation*, *Sampling*, *Balance*). All plots report the 95% stratified bootstrap confidence intervals over 3 random seeds for each experiment. Curves in the top report the inter-quartile mean (IQM). See [Gorsane et al., 2022, Agarwal et al., 2021] for more details on the reported metrics. Details and references for the algorithms used are available in Table 1.

A first set of benchmarks was run on the 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. 4. Individual task results are reported in Fig. 5. All the algorithms, models, and tasks were run using the default BenchMARL configuration, available in the `conf` folder. The experiment hyperparameters are available in the `fine_tuned/vmas` folder. The obtained results match the ones reported by Bou et al. [2024] and Bettini et al. [2022]. An interactive version of these results is available here.

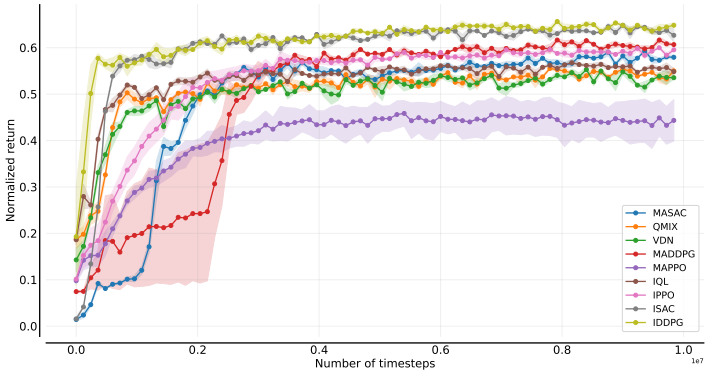
As we can see from the general plots in Fig. 4 (aggregated over all tasks), the Multi-Agent (MA) versions of all algorithms (i.e., MASAC, MADDPG, MAPPO) obtain the best overall performance. This is because these algorithms benefit from centralized critics during training, which allows the critic to condition on the global state instead of using just local information. Q-Learning algorithms (i.e., IQL, VDN, QMIX) perform suboptimally compared to the actor-critic ones. This might be due to the fact that these approaches are trained on a version of the tasks with discrete actions, which might impede performance given the continuous multi-robot control nature of these tasks.

5 Conclusion

In this paper we present BenchMARL, the first MARL benchmarking library with the goal of enabling standardization and reproducibility in the field. BenchMARL focuses on high-level structuring of configuration and reporting while using low-level benchmarked RL implementations from TorchRL. Thanks to this, it provides a lightweight and easy-to-use MARL training library that has already proved successful in multi-robot learning and zero-shot deployment in the real world [Blumenkamp et al., 2024]. 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.

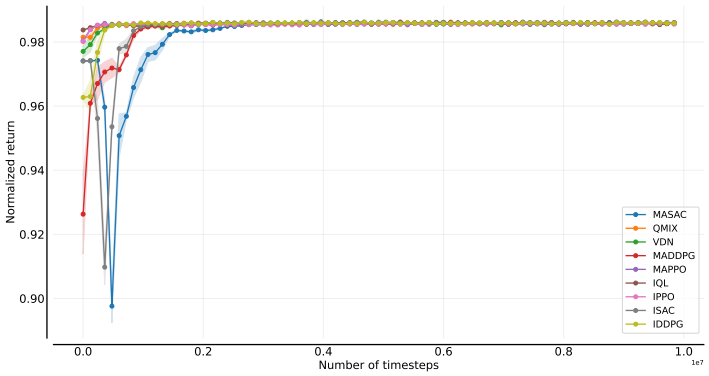


(a) *Sampling* task.

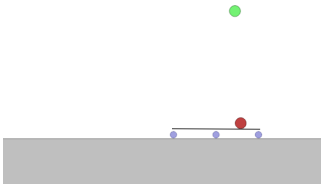


(b) *Sampling* reward curves.

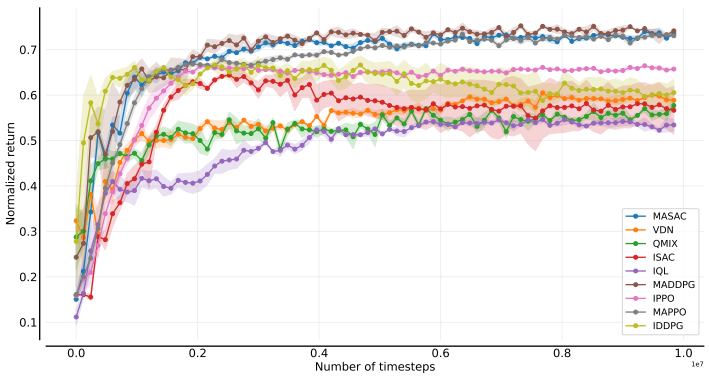
(c) *Navigation* task.



(d) *Navigation* reward curves.



(e) *Balance* task.



(f) *Balance* reward curves.

Figure 5: The sample efficiency curves for all BenchMARL algorithms over the three VMAS tasks analyzed. We report the inter-quartile mean (IQM) with 95% stratified bootstrap confidence intervals over 3 random seeds for each experiment. Details and references for the algorithms used are available in Table 1.

Acknowledgments

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.

References

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. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=QxItoEAVMb.

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

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.

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.

Andy B Yoo, Morris A Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*, pages 44–60. Springer, 2003.

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.

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.

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.

Siyi Hu, Yifan Zhong, Minquan Gao, Weixun Wang, Hao Dong, Xiaodan Liang, Zhihui Li, Xiaojun Chang, and Yaodong Yang. Marllib: A scalable and efficient multi-agent reinforcement learning library. *Journal of Machine Learning Research*, 24(315):1–23, 2023.

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.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Ruan de Kock, Omayma Mahjoub, Sasha Abramowitz, Wiem Khlifi, Callum Rhys Tilbury, Claude Formanek, Andries Smit, and Arnu Pretorius. Mava: a research library for distributed multi-agent reinforcement learning in jax. *arXiv preprint arXiv:2107.01460*, 2021.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments and algorithms in jax. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pages 2444–2446, 2024.

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.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043, 2021.

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.

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.

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.

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.

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.

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.

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.

Joel Z. Leibo, Edgar Dué nez Guzmán, Alexander Sasha Vezhnevets, John P. Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charles Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. PMLR, 2021. doi: 10.48550/arXiv.2107.06857. URL `https://doi.org/10.48550/arXiv.2107.06857`.

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.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

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

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 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.

Jan Blumenkamp, Ajay Shankar, Matteo Bettini, Joshua Bird, and Amanda Prorok. The cambridge robomaster: An agile multi-robot research platform. *arXiv preprint arXiv:2405.02198*, 2024.