BenchMARL: Benchmarking Multi-Agent Reinforcement Learning

Matteo Bettini*,1
Amanda Prorok1
Vincent Moens2

* Work done during an internship at PyTorch, Meta.
1 Department of Computer Science and Technology, University of Cambridge, United Kingdom.
2 PyTorch Team, Meta.

Editor: Joaquin Vanschoren

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., 2024) 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 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.
Bettini 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 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 community standards and tools, with new libraries being frequently introduced, each focusing on specific algorithms, environments, or models. Popular examples are PyMARL (Samvelyan et al., 2019), PyMARL2 (Hu et al., 2021), and EPyMARL (Papoudakis et al., 2021), which are limited to environments with discrete action spaces and 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), which however lacks core features for state-of-the-art benchmarking, such as support for vectorized environments. 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 framework, these projects are important complementary tools to our work, which is in turn focused on PyTorch. The fragmentation of the MARL domain has recently led to a reproducibility crisis, highlighted by Gorsane et al. (2022). While the authors propose 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.

---

1. Based on the NeurIPS 21 Outstanding Paper by Agarwal et al. (2021)
Table 1: Algorithms in BenchMARL.

<table>
<thead>
<tr>
<th>Name</th>
<th>On/Off policy</th>
<th>Actor-critic</th>
<th>Full-observability in critic</th>
<th>Action compatibility</th>
<th>Probabilistic actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAPPO (Yu et al., 2021)</td>
<td>On</td>
<td>Yes</td>
<td>Yes</td>
<td>Continuous + Discrete</td>
<td>Yes</td>
</tr>
<tr>
<td>IPPO (de Witt et al., 2020)</td>
<td>On</td>
<td>Yes</td>
<td>No</td>
<td>Continuous + Discrete</td>
<td>Yes</td>
</tr>
<tr>
<td>MADDPG (Lowe et al., 2017)</td>
<td>Off</td>
<td>Yes</td>
<td>Yes</td>
<td>Continuous</td>
<td>No</td>
</tr>
<tr>
<td>IDDPG</td>
<td>Off</td>
<td>Yes</td>
<td>No</td>
<td>Continuous</td>
<td>No</td>
</tr>
<tr>
<td>MASAC</td>
<td>Off</td>
<td>Yes</td>
<td>Yes</td>
<td>Continuous + Discrete</td>
<td>Yes</td>
</tr>
<tr>
<td>ISAC</td>
<td>Off</td>
<td>Yes</td>
<td>No</td>
<td>Continuous + Discrete</td>
<td>Yes</td>
</tr>
<tr>
<td>QMIX (Hashid et al., 2018)</td>
<td>Off</td>
<td>No</td>
<td>NA</td>
<td>Discrete</td>
<td>No</td>
</tr>
<tr>
<td>VDN (Sunehag et al., 2017)</td>
<td>Off</td>
<td>No</td>
<td>NA</td>
<td>Discrete</td>
<td>No</td>
</tr>
<tr>
<td>IQL 1 (Tan, 1993)</td>
<td>Off</td>
<td>No</td>
<td>NA</td>
<td>Discrete</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2: Environments in BenchMARL. Renderings are shown in Fig. 4.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Tasks</th>
<th>Cooperation</th>
<th>Global state</th>
<th>Reward function</th>
<th>Action space</th>
<th>Vectorized</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMAS (Bettini et al., 2022)</td>
<td>27</td>
<td>Cooperative + Competitive</td>
<td>No</td>
<td>Shared + Independent + Global</td>
<td>Continuous + Discrete</td>
<td>Yes</td>
</tr>
<tr>
<td>SMAC Verus (Bans et al., 2022)</td>
<td>15</td>
<td>Cooperative</td>
<td>Yes</td>
<td>Global</td>
<td>Discrete</td>
<td>No</td>
</tr>
<tr>
<td>MPE (Lowe et al., 2017)</td>
<td>8</td>
<td>Cooperative + Competitive</td>
<td>Yes</td>
<td>Shared + Independent + Global</td>
<td>Continuous + Discrete</td>
<td>No</td>
</tr>
<tr>
<td>SSL (Gupta et al., 2017)</td>
<td>2</td>
<td>Cooperative</td>
<td>No</td>
<td>Shared</td>
<td>Continuous</td>
<td>No</td>
</tr>
<tr>
<td>Meltingpot (Leno et al., 2021)</td>
<td>49</td>
<td>Cooperative + Competitive</td>
<td>Yes</td>
<td>Independent</td>
<td>Discrete</td>
<td>No</td>
</tr>
</tbody>
</table>

2.1 Components

BenchMARL has a few core components, which correspond to classes in the codebase. Each component has a YAML configuration in a dedicated directory decoupled from the code.

**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. 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 command line or from a script.

**Algorithms.** Algorithms are an ensemble of components (e.g., loss, replay buffer) that determine the training strategy. In Tab. 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 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 Tab. 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 collection on GPU devices.

**Models.** BenchMARL models are blueprints designed to be used in multiple MARL contexts. They can be employed as policies or decentralized critics in Decentralized Training
Decentralized Execution (DTDE), or as centralized critics in Centralized Training Decentralized Execution (CTDE). 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 or chain them to build more complex networks. Currently BenchMARL supports Multi Layer Perceptron (MLP), GNN, Convolutional Neural Network (CNN), and Deep Sets (Zaheer et al., 2017) models.

2.2 Features

We present some of the core features that enable the libraries standardization goals.

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

3. Conclusion

In this paper we present BenchMARL, the first TorchRL-backed MARL benchmarking library with the goal of enabling standardization. BenchMARL focuses on high-level structuring of configuration and reporting while using low-level benchmarked RL implementations from TorchRL. The MARL community can take advantage of BenchMARL to easily compare and share 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.
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.

Appendix A. Experiment results

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 and as well as comparisons to the original papers.

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. 2. Individual task results are reported in Fig. 3. 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. An interactive version of these results is available at https://wandb.ai/matteobettini/benchmarl-public/reports/VMAS-Benchmarks--Vmlldzo1NzI4MDA5 The obtained results match the ones reported by Bou et al. (2024) and Bettini et al. (2022).

As we can see from the general plots in Fig. 2 (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 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 multi-robot control nature of these tasks.

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 Tab. 2.

References

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 Tab. 1.


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 Tab. 1.
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 Tab. 2.


