Mighty: A Comprehensive Tool for studying Generalization, Meta-RL and AutoRL

Aditya Mohan[*]
Institute of Artificial Intelligence
Leibniz University Hannover
a.mohan@ai.uni-hannover.de

Theresa Eimer[*]
Institute of Artificial Intelligence
Leibniz University Hannover

Carolin Benjamins
Institute of Artificial Intelligence
Leibniz University Hannover

Marius Lindauer
Institute of Artificial Intelligence
L3S Research Center
Leibniz University Hannover

André Biedenkapp
Machine Learning Lab
University of Freiburg

Abstract

Robust generalization, rapid adaptation, and automated tuning are essential for deploying reinforcement learning in real-world settings. However, research on these aspects remains scattered across non-standard codebases and custom orchestration scripts. We introduce *Mighty*, an open source library that unifies Contextual Generalization, Meta-RL, and AutoRL under a single modular interface. Mighty cleanly separates a configurable *Agent* - specified by its learning algorithm, model architecture, replay buffer, exploration strategy, and hyperparameters - from a configurable environment modeled as a *Contextual MDP* in which transitions, rewards, and initial states are governed by context parameters. This design decouples *inner-loop* weight updates from *outer-loop* adaptations, enabling users to compose, within one framework, *(i)* contextual generalization and curriculum methods (e.g. Unsupervised Environment Design), *(ii)* bi-level meta-learning (e.g. MAML, black-box strategies), and *(iii)* automated hyperparameter and architecture search (e.g. Bayesian optimization, evolutionary strategies, population-based training). In this paper, we present Mighty's design philosophy and core features and validate the ongoing base implementations on classic control and continuous control tasks. We hope that by providing a unified, modular interface Mighty will simplify experimentation and inspire further advances in robust, adaptable reinforcement learning. Mighty is available at `https://github.com/automl/Mighty`.

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful paradigm for training agents to make decisions in complex and dynamic environments. Despite impressive successes in domains such as games [Silver et al., 2016, Badia et al., 2020, Vasco et al., 2024] and robotics [Lee et al., 2020], standard RL algorithms frequently overfit to their training conditions and struggle to generalize to new tasks or variations in the environment [Benjamins et al., 2023, Kirk et al., 2023, Mohan et al., 2024]. Addressing this challenge requires methods that not only learn efficiently on a single task but also adapt rapidly to novel settings and automatically tune their own learning process.

Recent research has advanced in three complementary directions. *(i)* **Generalization in RL**, which trains agents over a distribution of environment variants - often modeled as contextual MDPs - to

[*]Equal contribution.

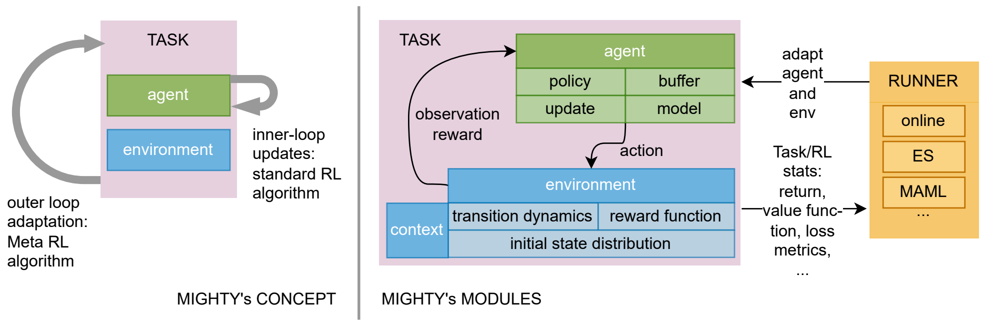

Figure 1: Overview of Mighty's concept and modules.

ensure robust transfer to unseen configurations [Benjamins et al., 2023, Cho et al., 2024]; *(ii)* **Meta-Reinforcement Learning** (Meta-RL), which learns across a family of related tasks so that agents can rapidly adapt to new tasks via bi-level optimization methods [Kaushik et al., 2020, Beck et al., 2023]; and *(iii)* **Automated Reinforcement Learning** (AutoRL), which integrates outer-loop optimization- e.g. Bayesian optimization, evolutionary strategies, or population-based training - directly into the RL pipeline to automate hyperparameter and architecture selection [Parker-Holder et al., 2022, Mohan et al., 2023, Eimer et al., 2023]. Although each paradigm has yielded powerful algorithms, researchers often find themselves integrating disparate libraries and custom scripts for environment generation, base RL training, and outer-loop optimization. This fragmentation imposes significant engineering overhead, slows iteration of new ideas, and hinders reproducible comparisons [Dizon-Paradis et al., 2024].

To address these challenges, we introduce *Mighty*: a unified modular library that streamlines the implementation, evaluation, and comparison of algorithms at the intersection of generalization, Meta-RL, and AutoRL. Mighty conceptualizes each task as the interaction of two core elements: the *agent* - specified by its policy and/or model parameterizations, exploration strategy, replay buffer, learning algorithm, and hyperparameters - and the *environment* -modeled as a contextual MDP [Hallak et al., 2015], where context parameters govern transitions, rewards, and initial states. Rather than scattering isolated scripts for curricula, adaptation, and tuning, Mighty enforces a clean separation along two axes of variation: (i) *Inner-loop updates*, which refine the agent's weights via typical RL training steps; and (ii) *Outer-loop adaptations*, which systematically adjust the agent's configuration, modify environmental contexts, or both. Through this separation, Mighty enables composing curriculum and context scheduling for robust generalization, bi-level or evolutionary Meta-RL for rapid adaptation, and automated hyperparameter optimization - all without ad-hoc orchestration code. An overview of our framework is shown in Figure 1.

In the following sections, we detail the core components and functionalities of Mighty, describe our experimental setup and results, and discuss potential future directions for research and development. Through this work, our goal is to provide the RL community with a powerful tool to enhance the generalizability of RL agents and inspire innovative approaches in Meta-RL.

2 Design Philosophy

Mighty aims to make generalization research, MetaRL, and AutoRL easier to implement and run. To achieve this goal, we focus on three core principles in designing Mighty: flexibility, integration with existing libraries, and CPU execution.

Flexibility. Research on generalization, MetaRL, and AutoRL currently takes many different forms: methods use black-box outer loops, add an additional inner loop relying on algorithm statistics, or directly work upon the environment. Consequently, we support the development of diverse methods instead of constraining them through library design. Therefore, we enable as many different ways of interacting with the core RL loop as possible. In particular, this also includes access to information on training stages such as transitions, predictions, networks, and environments to meta-methods. We prioritize such flexibility to enable all sorts of interaction patterns above, e.g., runtime efficiency, because we believe that a sufficient number of libraries focuses on runtime optimization. Instead, we aim to inspire the development of novel methods by providing an open and flexible platform on top of base RL algorithms.

Integrations. In order to achieve the desired level of flexibility inherent in Mighty, it is necessary to reimplement the foundational algorithms. However, to minimize unnecessary maintenance burdens, we aim to leverage existing libraries wherever possible to extend functionality. This involves the utilization of diverse environment libraries such as Gymnasium [Towers et al., 2024], Pufferlib [Suarez, 2025], and CARL [Benjamins et al., 2023] to facilitate capabilities such as parallel evaluation, run-time optimization, and modeling of intra-environment variations for enhanced generalization. We also rely on Hydra [Yadan, 2019] as a command-line interface which enables out-of-the-box parallelization in clusters and hyperparameter optimization. Lastly, our flexible interface allows for easy connection points with other libraries. We provide an example of such an integration in Mighty's evosax [Lange, 2022] runner: in less than 100 lines of code, we implement a runner class that can use any evosax algorithm for either hyperparameter optimization or policy optimization. These integrations let us focus on our core mission while still providing important quality of life features in Mighty.

Environment Optimization First. Much of machine learning relies on improving prediction efficiency, e.g. through optimal GPU usage, for improved runtimes. In part due to the introduction of JAX to the community [Lu et al., 2022, Park et al., 2025], RL is adopting a similar trend. While this is certainly important in many use cases with high compute cost, it also comes with implementation complexity, and many environments are still bound to the CPU only. Thus, we choose a different approach and focus on usability instead of speed for agent implementations and instead rely on vectorized CPU environments for speedups. Pufferlib [Suarez, 2025] specifically optimizes for most environment steps per second, allowing us to focus on flexibility instead of speed in Mighty itself. This allows Mighty consistently high training speeds even on CPUs for all but edge cases (e.g. extremely costly applications or fully sequential environment interactions). Therefore, we believe that this choice strikes a balance between efficient runtimes and flexibility in implementation in Mighty's target use cases.

3 Existing Tools for RL and Meta RL

The rapidly growing ecosystem of RL libraries spans diverse design philosophies - from low-level composability to turnkey baselines and massive-scale engines - making direct comparison and tool selection challenging. To clarify these trade-offs, we categorize existing frameworks into three archetypes: *Modular Research Frameworks*, *Monolithic Baselines*, and *Scalable Platforms*. This categorization highlights the trade-offs between composability, ease of use, scalability, and motivates the need for a unified interface that spans all three dimensions.

3.1 Modular Research Frameworks

Modular research frameworks expose the internal building blocks of an RL pipeline - networks, replay buffers, data collectors, and transforms - as standalone components that can be recombined to quickly prototype new algorithms. TorchRL [Bou et al., 2023] pioneered this approach in the PyTorch ecosystem, introducing the `TensorDict` abstraction to seamlessly pass the observed actions and rewards between modules. Tianshou [Weng et al., 2022] offers a similarly flexible design with separate `Policy`, `Collector`, and `Buffer` classes, enabling researchers to switch custom exploration strategies or data collection schemes with minimal boilerplate. While these libraries excel at inner loop algorithm development and fine-grained experimentation, they leave higher-order workflows - such as curriculum learning, meta-adaptation across tasks, and automated hyperparameter optimization - to external scripts or user-written loops. Mighty uses a similar modular architecture for inner-loop methods (`Policy`, `Model`, `Agent`, `Update`, `Exploration`), but in contrast, it builds on this composable foundation and adds outer loop constructs in the form of different `Runners` that orchestrate variation of the environment, metalearning updates and hyperparameter optimization within the same framework.

3.2 Monolithic Baselines

Monolithic baselines such as stable baselines3 (SB3) [Raffin et al., 2021] and CleanRL/PureJaxRL [Huang et al., 2022, Lu et al., 2022] prioritize ease of use and reproducibility. With just a few lines of code, users can train state-of-the-art algorithms (e.g., PPO [Schulman et al., 2017], DQN [Silver et al., 2016], SAC [Haarnoja et al., 2018]) on standard benchmarks, benefiting from sensible defaults and integrated logging. However, this simplicity comes at the cost of extensibility: SB3's algorithms hide most of the training loop behind a single `learn()` call, and CleanRL's single file scripts are not designed for import or extension. Neither framework provides built-in hooks for systematic curriculum scheduling, meta-level adaptation, or inner-loop hyperparameter tuning. Mighty complements these baselines by preserving their turnkey usability for standard experiments while exposing the same high-level API to more advanced outer loop workflows, so that curriculum, meta-learning, and AutoRL can be applied with equal ease.

3.3 Scalable Platforms

Scalable platforms such as RLlib [Liang et al., 2018, Wu et al., 2021] and STOIX [Toledo, 2024] focus on maximizing throughput and supporting distributed execution. RLlib integrates tightly with the Ray ecosystem to run thousands of environment replicas across CPU clusters or multi-GPU setups, and offers seamless Ray Tune integration for large-scale hyperparameter sweeps. STOIX uses JAX's `jit` and `pmap` primitives to compile both agent and environment code for extreme speed, even in single-file examples. Although these systems shine when running large experiments, their APIs do not natively unify component modularity with built-in meta-learning or curriculum design. RLlib's meta-learning methods must be configured through complex JSON schemas, and STOIX's examples are not structured as importable libraries. Mighty occupies the middle ground, offering efficient single-node performance via PyTorch, straightforward multicore environment parallelism, and a modular interface, while also providing first-class support for metaadaptation, curriculum scheduling, and automated optimization, all within the same cohesive framework.

4 Features and Usage

Mighty's main goal is to accelerate experimentation by combining an intuitive interface with robust, modular components. This design enables researchers to focus on advancing learning strategies and improving generalization capabilities instead of on low-level algorithm code. Here we highlight some of Mighty's features that help achieve this goal: (i) our user interface, (ii) the modular algorithms, and (iii) the flexible integration of meta-methods.

4.1 User Interface

Mighty is designed with user-friendliness and flexibility as top priorities. Thus, the interface to the user and interface opportunities for other libraries are essential for Mighty. Choosing Hydra [Yadan, 2019] for the command line accomplishes both: structured configuration files make it possible to expose all relevant training details in Mighty without overwhelming new users (see Figure 2). On the other hand, it integrates Mighty into Hydra's plugin offerings. This is particularly important for experimentation on clusters and hyperparameter optimization, both of which are plug-and-play for Mighty and avoid brittle dependency chains. Our examples show that with only a few lines added to the configuration file, users can run parallel hyperparameter optimization runs or distribute runs across seeds and environments on a HPC cluster. Additional helpers, such as live plotting of current results and an integration with Weights & Biases [Biewald, 2020], help users keep track of these data and make debugging and analysis simple.

Furthermore, Mighty is extensible, meaning that each component in Mighty's algorithms can be subclassed, and thus replaced in the configuration by a local component. Therefore, it is not necessary to clone the entire Mighty code to add a new functionality by hacking the scripts: users can choose what they want to add, e.g., a replay buffer, and implement a new subclass of `MightyReplay`. The target algorithm can then be configured to use this new replay buffer by specifying the corresponding import path. In this case, the user can run the new replay buffer class without touching the algorithm loop itself. This keeps the codebases *small, easy to maintain, and focused on the research goal*.

Base Configuration With Defaults

```yaml
defaults:
  - _self_
  - algorithm: ppo
  - environment: pufferlib_ocean/bandit
  - search_space: dqn_gym_classic
  - override hydra/job_logging: colorlog
  - override hydra/hydra_logging: colorlog
  - override hydra/help: mighty_help

runner: standard
debug: false
seed: 0
output_dir: runs
wandb_project: null
tensorboard_file: null
experiment_name: mighty_experiment

algorithm_kwargs: {}

eval_every_n_steps: 5e3
n_episodes_eval: 10
checkpoint: null
save_model_every_n_steps: 5e5

hydra:
  run:
    dir: ${output_dir}/${experiment_name}_${seed}
  sweep:
    dir: ${output_dir}/${experiment_name}_${seed}
```

Example Algorithm Configuration

```yaml
# @package _global_
algorithm: PPO

algorithm_kwargs:
    rollout_buffer_class:
        _target_: mighty.mighty_replay.MightyRolloutBuffer
    rollout_buffer_kwargs:
        buffer_size: 256
        gamma: 0.98
        gae_lambda: 0.8

    learning_rate:  3e-4
    batch_size: 32
    gamma: 0.99
    ppo_clip: 0.2
    value_loss_coef: 0.5
    entropy_coef: 0.0
    max_grad_norm: 0.5

    n_epochs: 20
    minibatch_size: 256
    kl_target: 0.01
    use_value_clip: True
    value_clip_eps: 0.2

    policy_class: mighty.mighty_exploration.StochasticPolicy
    policy_kwargs:
        entropy_coefficient: 0.0
```

Figure 2: Configuration examples: **Left** a base config file with defaults to configure only necessary information; **Right** configuration for PPO with important training details available at a glance.

```python
class DoubleQLearning(QLearning):
    """Double Q-learning update."""

    def get_targets(self, batch, q_net, target_net):
        """Compute targets with target network"""
        next_actions = q_net(batch.next_obs)
            .argmax(dim=1)

        target_qs = target_net(batch.next_obs)
            .gather(1, next_actions)


        targets = batch.rewards + (1-batch.dones) * self.gamma * target_qs
        q_vals = q_net(batch.observations)
            .gather(1, batch.actions)
        return q_vals, targets
```

Figure 3: Extending an update class in Mighty: double Q-learning inherits the original Q-learning update and only overwrites the target generation. No other elements of DQN need to be altered in any way as long as this class is selected via the command line.

Let us look at an example. To integrate domain randomization [Tobin et al., 2017] through Syllabus [Sullivan et al., 2025], we need around 100 lines of code each to interface Syllabus and build a custom task wrapper. With our Mighty project template (containing basic code structure, runscripts and configuration files) as a base, *less than* 200 *lines of Python code and three configuration files* are enough for a full evaluation, including hyperparameter optimization and cluster deployment (see the example project repository including results).

4.2 Agent Framework

Mighty implements three base RL algorithms: DQN [Mnih et al., 2015], SAC [Haarnoja et al., 2018], and PPO [Schulman et al., 2017]. Each of these agents is composed of four core components which can easily be extended:

```python
    def run(self) -> Tuple[Dict, Dict]:
        es_state = self.es.initialize(self.rng)
        for _ in range(self.iterations):
            # Get population
            x, es_state = self.es.ask(es_state)
            eval_rewards = []
            # Evaluate individuals
            for individual in x:
                self.apply_parameters(individual)
                eval_results = self.evaluate()
                eval_rewards.append(eval_results["mean_eval_reward"])
            # Feedback to ES
            fitness = self.fit_shaper.apply(x, eval_rewards)
            es_state = self.es.tell(x, fitness, es_state)
        eval_results = self.evaluate()
        return {"step": self.iterations}, eval_results
```

Figure 4: Evosax run function for policy search. Agent-related functionality has a unified interface, so the runner can be focused on the evosax functionality.

1. **Exploration Strategy:** Defines how the agent's action predictions are transmitted to the environment, e.g., by adding random sampling on top of the predictions.
2. **Replay Buffer:** A storage mechanism that records past interactions. Takes the form of a replay buffer that stores single transitions or a rollout buffer that stores full trajectories.
3. **Update Function:** The algorithmic core responsible for updating the agent policy.
4. **Model parametrization:** The underlying neural network that approximates the policy or value function(s).

This modular approach enables users to seamlessly substitute or extend individual components according to their research needs. Figure 3, for example, shows the double Q-learning update, which is built as an extension of the Q-learning update class. As these four components contain much of the algorithm functionality, we expect most user needs in the realm of generalization, MetaRL, and AutoRL research to be met without the need to implement a fully new algorithm class or review the interaction loop.

4.3 Meta-Learning Framework

The core strength of Mighty lies in its capability to incorporate meta-methods. There are two types of interaction that can be combined: *outer-loop runners* and *inner-loop meta-components*.

Runners interact with RL tasks directly. They can alter the agent or environment and execute (partial) runs. They also have access to training results and artifacts such as evaluation rewards, training environments, or policy parameters. This enables hyperparameter optimization, policy search using evolutionary algorithms (as seen in our evosax [Lange, 2022] runner), and even complex MetaRL methods like e.g. MAML [Finn et al., 2017] that adapt both environment and policy jointly. Moreover, the structure of these runners is very simple, one only needs to implement a *run* method (see Figure 4 for a simple online runner and the evosax runner). This simplicity makes runners a great way to integrate other learning paradigms or packages.

Meta-Components interact with the algorithm in the inner loop in a single lifetime. They have access to six interaction hooks (pre- and post-steps, episodes, and updates) and access to all available training information. This includes transitions, losses, predictions, update batches, and more. It enables meta-components to serve as base classes for different meta-methods such as curriculum generators [Mehta et al., 2020], intrinsic rewards [Yuan, 2022], sparsity and activations [Ma et al., 2025, Delfosse et al., 2024, Loni et al., 2023], and hyperparameter schedules [Eimer et al., 2023, Mohan et al., 2023]. Like runners, they only need to implement their needed hooks and are thus extremely flexible. They can also be stacked to enable research on combinations of different learning paradigms.

While meta-components are in principle single-lifetime approaches, they can be extended to multiple runtimes via runners. Therefore, runners can evolve meta-components across multiple trials, thus enabling a plethora of interactions with the core RL loop. Due to their modularity and flexibility, such meta-methods can be implemented for and applied to all compatible base algorithms. Runners and meta-components can be freely combined and stacked, and we can do so with little code overhead. This approach is *unique* to Mighty and will enable new directions in RL by letting researchers work in the intersections of different meta-research directions.

4.4 Currently Implemented Methods

Mighty is primarily a platform for implementing new research, but it comes with several built-in options as seen in Table 1 and our documentation. We ensured that the added functionality demonstrates extensions for each of Mighty's components. The ϵz-greedy [Dabney et al., 2021] exploration, prioritized replay buffer [Schaul. et al., 2016], and DDQN [van Hasselt et al., 2016] update combined with network variations each expand upon the core agent components. The evosax runner shows how to access different elements of the training pipeline and execute runs, while the different meta-components give examples of online interactions with hyperparameters (cosine annealing; Loshchilov and Hutter [2017]), transitions & training batches (RND and NovelD; Burda et al. [2019], Zhang et al. [2021]) and contextual environments (PLR and SPaCE; Jiang et al. [2021], Eimer et al. [2021a]). Thus, most users will have a template for the interaction they are implementing while having access to a robust set of core algorithms. Mighty also contains examples for environments from Gymnasium [Towers et al., 2024], Pufferlib [Suarez, 2025], CARL [Benjamins et al., 2023] and DACBench [Eimer et al., 2021b], including the use of context distributions.

Table 1: Current scope of Mighty.

Networks	Algorithms	Policies	Environments	Meta	Runners
MLP (flexible)	DQN	ϵ-greedy	Gymnasium	PLR	Online RL
CNN (flexible)	DDQN	ϵz-greedy	Pufferlib	SPaCE	Evosax
ResNet	PPO	stochastic	CARL	RND/NovelD	
Combinations	SAC		DACBench	Cosine Annealing LR	

5 Empirical Validation

We validate our implementations by comparing them with OpenRL benchmark results (where available) [Huang et al., 2024]. Our aim is not to outperform existing baselines, but to demonstrate that Mighty achieves comparable performance at similar training budgets. Table 2 reports the number of training steps, average wall clock time, and comparison of the final results between our implementations and the OpenRL reference values. The trends broadly align: PPO and DQN on CartPole closely track OpenRL, and PPO on MountainCar reproduces the expected -200 plateau. Deviations appear where exploration and continuous-control dynamics matter more: DQN on MountainCar remains at -200 on our runs while OpenRL occasionally escapes. SAC in Walker2D and HalfCheetah remains close to the mean performance reported by OpenRL, and within the variance of their performance across seeds. In general, the results demonstrate that Mighty's implementations reproduce the results of established baselines, both in sample efficiency and runtime.

Table 2: Return across 5 seeds, runtime on a MacBook Pro with an Apple M2 Max chip.

Algo.	Environment	Steps	Time (min)	Final Return		OpenRL Return	
DQN	MountainCar	5×10^5	51,1	-200.00	± 0.00	-189.92	± 11.00
DQN	CartPole	5×10^5	60.41	486.40	± 30.77	499.92	± 0.00
PPO	MountainCar	5×10^5	3.03	-200.00	± 0.00	-200.00	± 0.00
PPO	CartPole	5×10^5	3.67	479.80	± 17.21	487.48	± 6.79
PPO	Pendulum	5×10^5	4.05	-255.00	± 11.14	–	-
SAC	Walker2D	10^6	353.13	4478.67	± 689.22	4471.15	± 1896.34
SAC	HalfCheetah	10^6	302.53	10588.34	± 874.19	10958.60	± 1335.62

6 Conclusion

In this work, we have introduced Mighty, a unified modular library designed to streamline and accelerate research at the intersection of contextual generalization, Meta-RL, and AutoRL. By treating inner-loop learning (i.e., standard policy updates and environment interactions) and outer-loop adaptation (i.e., curriculum or context scheduling, meta-parameter updates, and hyperparameter search) as distinct, first-class abstractions, Mighty eliminates the need for ad-hoc orchestration scripts. This clean separation not only reduces engineering overhead—allowing researchers to focus on algorithmic innovation rather than low-level integration—but also fosters reproducibility, since all components (agents, environments, meta-components, and runners) can be combined in a standardized way without custom glue code. Thus, we believe that Mighty will empower the community to iterate more rapidly on new algorithms and advance the state of robust, adaptable RL.

Acknowledgements

Aditya Mohan acknowledges funding by the German Research Foundation (DFG) under LI 2801/10-1. Theresa Eimer acknowledges funding by the German Research Foundation (DFG) under LI 2801/7-1. André Biedenkapp acknowledges funding through the research network “Responsive and Scalable Learning for Robots Assisting Humans” (ReScaLe) of the University of Freiburg. The ReScaLe project is funded by the Carl Zeiss Foundation.

References

A. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. In H. Daume III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning (ICML’20)*, volume 98. Proceedings of Machine Learning Research, 2020.

J. Beck, R. Vuorio, E. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson. A survey of meta-reinforcement learning. *CoRR*, abs/2301.08028, 2023.

C. Benjamins, T. Eimer, F. Schubert, A. Mohan, S. Döhler, A. Biedenkapp, B. Rosenhan, F. Hutter, and M. Lindauer. Contextualize me – the case for context in reinforcement learning. *Transactions on Machine Learning Research*, 2023.

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

A. Bou, M. Bettini, S. Dittert, V. Kumar, S. Sodhani, X. Yang, G. D. Fabritiis, and V. Moens. Torchrl: A data-driven decision-making library for pytorch, 2023.

Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. Efros. Large-scale study of curiosity-driven learning. In *The Seventh International Conference on Learning Representations (ICLR’19)*. ICLR, 2019. Published online: iclr.cc.

J. Cho, V. Jayawardana, S. Li, and C. Wu. Model-based transfer learning for contextual reinforcement learning. In *Proceedings of the 38th International Conference on Advances in Neural Information Processing Systems (NeurIPS’24)*. Curran Associates, 2024.

W. Dabney, G. Ostrovski, and A. Barreto. Temporally-extended ϵ-greedy exploration. In *The Ninth International Conference on Learning Representations (ICLR’21)*. ICLR, 2021. Published online: iclr.cc.

Q. Delfosse, P. Schramowski, M. Mundt, A. Molina, and K. Kersting. Adaptive rational activations to boost deep reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=g90ysX1sVs.

O. Dizon-Paradis, S. Wormald, D. Capecci, A. Bhandarkar, and D. Woodard. Resource usage evaluation of discrete model-free deep reinforcement learning algorithms. *Reinforcement Learning Journal*, 2024.

T. Eimer, A. Biedenkapp, F. Hutter, and M. Lindauer. Self-paced context evaluation for contextual reinforcement learning. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML’21)*, volume 139 of *Proceedings of Machine Learning Research*, pages 2948–2958. PMLR, 2021a.

T. Eimer, A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, and M. Lindauer. DACBench: A benchmark library for dynamic algorithm configuration. In Z. Zhou, editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI'21)*, pages 1668–1674. ijcai.org, 2021b.

T. Eimer, M. Lindauer, and R. Raileanu. Hyperparameters in reinforcement learning and how to tune them. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, volume 202 of *Proceedings of Machine Learning Research*. PMLR, 2023.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, volume 70, pages 1126–1135. Proceedings of Machine Learning Research, 2017.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80. Proceedings of Machine Learning Research, 2018.

A. Hallak, D. D. Castro, and S. Mannor. Contextual markov decision processes. *arXiv:1502.02259 [stat.ML]*, 2015.

S. Huang, R. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

S. Huang, Q. Gallouédec, F. Felten, A. Raffin, R. F. J. Dossa, Y. Zhao, R. Sullivan, V. Makoviychuk, D. Makoviichuk, M. H. Danesh, et al. Open rl benchmark: Comprehensive tracked experiments for reinforcement learning. *arXiv preprint arXiv:2402.03046*, 2024.

M. Jiang, E. Grefenstette, and T. Rocktäschel. Prioritized level replay. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning (ICML'21)*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2021.

R. Kaushik, T. Anne, and J. Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS'20)*, 2020.

R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research (JAIR)*, 76:201–264, 2023.

R. T. Lange. evosax: Jax-based evolution strategies. *arXiv preprint arXiv:2212.04180*, 2022.

J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science in Robotics*, 5, 2020.

E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80. Proceedings of Machine Learning Research, 2018.

M. Loni, A. Mohan, M. Asadi, and M. Lindauer. Learning activation functions for sparse neural networks. In A. Faust, C. White, F. Hutter, R. Garnett, and J. Gardner, editors, *Proceedings of the Second International Conference on Automated Machine Learning*. Proceedings of Machine Learning Research, 2023.

I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *The Fifth International Conference on Learning Representations (ICLR'17)*. ICLR, 2017. Published online: `iclr.cc`.

C. Lu, J. Kuba, A. Letcher, L. Metz, C. de Witt, and J. Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 2022.

G. Ma, L. Li, Z. Wang, L. Shen, P. Bacon, and D. Tao. Network sparsity unlocks the scaling potential of deep reinforcement learning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=mIomqOskaa.

B. Mehta, T. Deleu, S. Raparthy, C. Pal, and L. Paull. Curriculum in gradient-based meta-reinforcement learning. *arXiv preprint arXiv:2002.07956*, 2020.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

A. Mohan, C. Benjamins, K. Wienecke, A. Dockhorn, and M. Lindauer. Autorl hyperparameter landscapes. In A. Faust, C. White, F. Hutter, R. Garnett, and J. Gardner, editors, *Proceedings of the Second International Conference on Automated Machine Learning*. Proceedings of Machine Learning Research, 2023.

A. Mohan, A. Zhang, and M. Lindauer. Structure in deep reinforcement learning: A survey and open problems. *Journal of Artificial Intelligence Research*, 79, 2024.

S. Park, K. Frans, B. Eysenbach, and S. Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *The Thirteenth International Conference on Learning Representations (ICLR'25)*. ICLR, 2025. Published online: iclr.cc.

J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, F. Hutter, and M. Lindauer. Automated reinforcement learning (AutoRL): A survey and open problems. *Journal of Artificial Intelligence Research (JAIR)*, 74:517–568, 2022.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22 (268):1–8, 2021.

T. Schaul., J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *The Fourth International Conference on Learning Representations (ICLR'16)*. ICLR, 2016. Published online: iclr.cc.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347 [cs.LG]*, 2017.

D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

J. Suarez. Pufferlib 2.0: Reinforcement learning at 1m steps/s. *Reinforcement Learning Journal*, 2025.

R. Sullivan, R. Pégoud, A. Rahmen, X. Yang, J. Huang, A. Verma, N. Mitra, and J. Dickerson. Syllabus: Portable curricula for reinforcement learning agents. *RLJ*, 2025.

J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pages 23–30. IEEE, 2017.

E. Toledo. Stoix: Distributed single-agent reinforcement learning end-to-end in jax, Apr. 2024. URL https://github.com/EdanToledo/Stoix.

M. Towers, A. Kwiatkowski, J. Terry, J. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In D. Schuurmans and M. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16)*, page 2094–2100. AAAI Press, 2016.

M. Vasco, T. Seno, K.Kawamoto, K. Subramanian, P. Wurman, and P. Stone. A super-human vision-based reinforcement learning agent for autonomous racing in gran turismo. *RLJ*, 4:1674–1710, 2024.

J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022. URL http://jmlr.org/papers/v23/21-1127.html.

Z. Wu, E. Liang, M. Luo, S. Mika, J. Gonzalez, and I. Stoica. RLlib flow: Distributed reinforcement learning is a dataflow problem. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Curran Associates, 2021.

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

M. Yuan. Intrinsically-motivated reinforcement learning: A brief introduction. *arXiv preprint arXiv:2203.02298*, 2022.

T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. Gonzalez, and Y. Tian. Noveld: A simple yet effective exploration criterion. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Curran Associates, 2021.