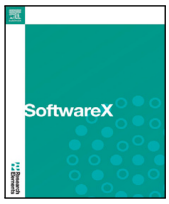
SoftwareX 29 (2025) 102064

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx



Original software publication

Scilab-RL: A software framework for efficient reinforcement learning and cognitive modeling research

Jan Benad *, Frank Röder, Manfred Eppe

Institute for Data Science Foundations, Hamburg University of Technology, Hamburg, Germany

ARTICLE INFO

Keywords:
Reinforcement learning
Cognitive modeling
Robotics
Python

ABSTRACT

One problem with researching cognitive modeling and reinforcement learning (RL) is that researchers spend too much time on setting up an appropriate computational framework for their experiments. Many open source implementations of current RL algorithms exist, but there is a lack of a modular suite of tools combining different robotic simulators and platforms, data visualization, hyperparameter optimization, and baseline experiments. To address this problem, we present Scilab-RL, a software framework for efficient research in cognitive modeling and reinforcement learning for robotic agents. The framework focuses on goal-conditioned reinforcement learning using Stable Baselines 3, CleanRL and the Gymnasium interface. It enables native possibilities for experiment visualizations and hyperparameter optimization. We describe how these features enable researchers to conduct experiments with minimal time effort, thus maximizing research output.

Code metadata

Current code version	v0.3.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00284
Legal Code License	MIT
Code versioning system used	Git
Software code languages, tools, and services used	Python, Shell
Compilation requirements, operating environments & dependencies	See requirements.txt, mujoco, PyTorch
If available, link to developer documentation/manual	https://scilab-rl.github.io/Scilab-RL/
Support email for questions	https://github.com/Scilab-RL/Scilab-RL/issues

1. Motivation and significance

Breakthroughs such as AlphaGo [1], solving Atari games [2], and most recently, the discovery of new approaches to faster matrix multiplication algorithms [3] are contributing to the growing popularity of reinforcement learning (RL). The development of new RL methods involves an iterative process of repeated design, testing, and analysis. To accelerate this process, researchers need a framework that includes a variety of robust RL algorithms and environments for benchmarking, and also possibilities for visualization, hyperparameter optimization and testing. Developing an integrated, robust workflow where all these tools synergize with each other is time-consuming and distracts from the actual research question in mind. Interdisciplinary researchers that are non-native programmers suffer most from this problem, even though the field of artificial intelligence can benefit tremendously from theories developed in for example psychology or cognitive science.

To address this problem, there exist computational frameworks for the rapid prototyping of new RL approaches and methods. As examples, consider SpinningUp [4], CleanRL [5], Garage [6], RLlib [7], Tianshou [8] or RL Baselines3 Zoo [9]. Some of them can be considered a very valuable entry point into RL, as their codebase is lightweight, concise and understandable [5,8] or their documentation itself can be considered as an educational resource to learn about RL [4]. Their strengths lie, for example, in supporting a wide range of algorithms [5–9], in performance [7], or in reproducibility [6]. However, some are missing a direct way of integrating supportive tools e.g. for hyperparameter optimization [6,8]. While options to support cloud-based experiment tracking and metric logging are quite common among this kind of frameworks [4–9], an additional aspect that we have found to

* Corresponding author.
E-mail addresses: jan.benad@tuhh.de (Jan Benad), frank.roeder@tuhh.de (Frank Röder), manfred.eppe@tuhh.de (Manfred Eppe).

https://doi.org/10.1016/j.softx.2025.102064
Received 28 April 2023; Received in revised form 10 January 2025; Accepted 19 January 2025
Available online 31 January 2025

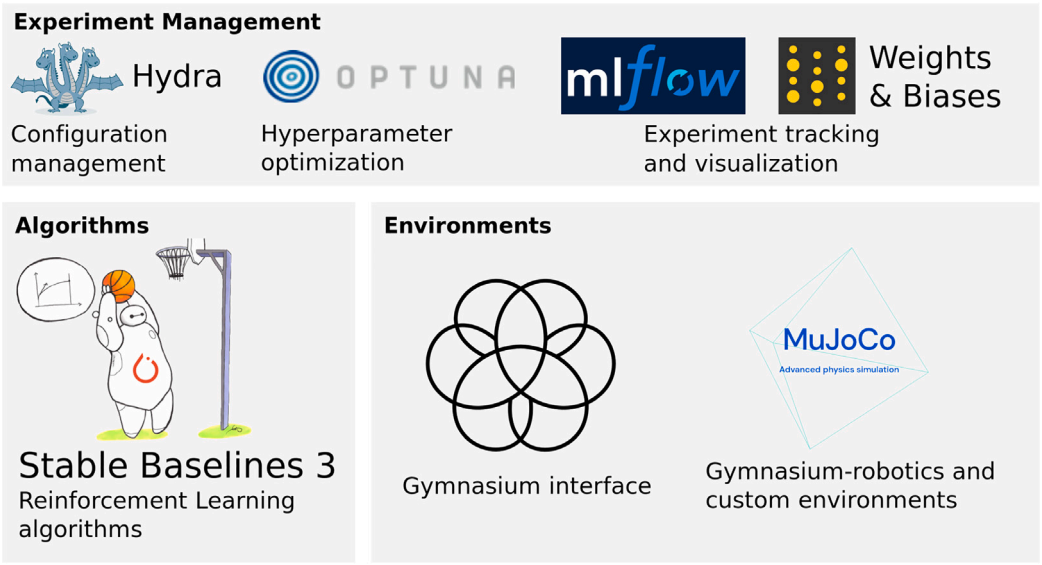


Fig. 1. An overview of the tools used in Scilab-RL.

be very valuable for RL development, but that to the best of our knowledge we could not find anywhere else, is the ability to observe metrics such as Q-values or rewards in parallel with the rendering of the environment. Another distinct feature compared to other frameworks [4–7] is the emphasis on and support for goal-conditioned RL.

In that regard, we propose Scilab-RL, a robust framework for efficient experimenting in RL. The framework is especially addressed to new researchers in the fields of RL and cognitive modeling and aimed at providing an easier access to computational modeling. In the context of robotic tasks we focus on methods that avoid reward-shaping, by using goal-conditioned reinforcement learning [10] and hindsight experience replay [11] instead. To assist in following the manuscript, we provide an easy access to (goal-conditioned) reinforcement learning in the Appendix A.

A core feature is the aforementioned visualization of metrics during runtime, as illustrated in Fig. 2. Overall, our framework involves the following features:

(a) A standardized sense-act interface based on Gymnasium [12], using the robotic simulator MuJoCo [13].
(b) A hyperparameter optimization engine based on Optuna [14].
(c) Implementations of the currently most important RL algorithms based on Stable Baselines 3 [15] and CleanRL [5].
(d) Multiple data visualization methods, including experiment monitoring using MLFlow [16] or Weights & Biases [17] and online visualization of metrics, like Q-values or intrinsic rewards.
(e) Testing scripts for continuous integration and delivery.

To obtain Scilab-RL, visit our GitHub repository (see Code metadata table). The tool is designed to be executed from the command line, with configurations specified in dedicated setting files. After running an experiment, the results can be analyzed using the integrated visualization capability or by linking to external visualization tools.

2. Software description

2.1. Software architecture

The core of Scilab-RL, which is written in Python, leverages YAML files to configure the algorithms and shell scripts to define the overall setup and testing procedures. It integrates RL algorithms from Stable Baselines 3 [15] and CleanRL [5], allowing straightforward pairing with the Gymnasium interface [12] for environment interactions. Furthermore, the robotic simulator MuJoCo [13] defines the framework's wide variety of mostly goal-conditioned benchmark tasks.

The framework incorporates specialized tools and customized functionalities to facilitate intuitive experiment management and evaluation. These include configuration management, hyperparameter optimization, experimental tracking, and visualization. Fig. 1 and Section 2.2 provide an overview of the Scilab-RL components.

2.2. Software functionalities

The following sections present the core functionalities. However, we recommend users to examine our dedicated Wiki[1] for additional tutorials, examples, and a detailed documentation.

2.2.1. RL algorithm development

Research on RL and cognitive modeling involves testing hypotheses by implementing new or modified RL algorithms. With our Stable Baselines 3 and CleanRL integration, this becomes straightforward using the Scilab-RL framework. The only requirement is that the developed algorithms follow the sense-act interface based on Gymnasium. The pivotal benefits that we offer, are the integrated data visualization, hyperparameter optimization, online rendering and tests.

2.2.2. Environment development

Our framework involves a variety of existing RL benchmark environments, particularly environments with sparse rewards for goal-conditioned RL to avoid reward-shaping. Environments from Gymnasium-Robotics and similar custom robotic tasks are directly usable [18]. However, it is often important for a research question to develop new environments. Using Scilab-RL speeds up this task because developers do not need to spend time on implementing RL algorithms as they are already integrated and ready-to-use. The only requirement for the environment development is, again, that it follows the sense-act interface based on Gymnasium.

[1] https://scilab-rl.github.io/Scilab-RL/

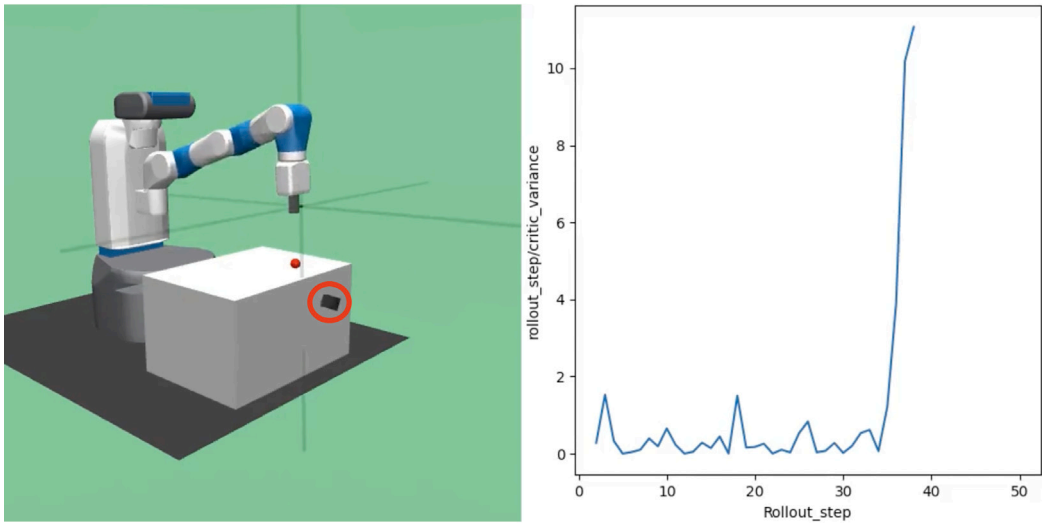


Fig. 2. Online rendering capabilities. Example using the MuJoCo FetchPush environment. The left side shows the rendered environment visualization. User-defined metric(s) are shown on the right. For this illustrative example, we track the variance of critics at each episode step. When the block falls off the table, the variance increases rapidly, indicating a rare event. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.2.3. Visualization

Developing and debugging new algorithms or environments can be quite challenging and time-consuming. Here, the visualization capability can be of great importance, e.g. for discovering functional flaws early on. We feature two kinds of data visualization. The first kind of visualization is common in similar RL-focused frameworks. It involves the cloud-based logging and tracking of metrics using MLFlow [16] and Weights & Biases [17]. This enables researchers to review ongoing experiments. For example, one can assess the learning performance during hyperparameter optimization, even when experiments are running on different machines.

The second kind of visualization is usually not available in similar frameworks, but it can be of immediate help: the online visualization of user-defined metrics in sync with the rendering provided by the respective environment. Developers can define and visualize any number of online metrics at runtime. In Section 3, we use the online metric visualization to make sense of the agent's behavior and thus verify a proposed hypothesis.

2.2.4. Hyperparameter optimization

Reinforcement learning typically involves many hyperparameters that are hard to manage and optimize. To address this issue, we use Hydra [19], a robust and versatile hyperparameter management system that connects seamlessly with hyperparameter optimization frameworks such as Optuna [14]. Default parameter configurations are stored in corresponding YAML files and are tied to an algorithm or an algorithm–environment combination. This makes it especially useful if different environment require different parameters. Furthermore, we allow alternating or adding individual parameter values.

2.2.5. Tests

Testing is important for continuous integration and delivery pipe lines. For example, when underlying packages receive an update, or a common utility function changes in the code, Scilab-RL offers performance and smoke tests to verify these. Smoke tests check the pure functionality, whether everything within the framework still runs. The performance tests verify that our RL algorithm's performance are not affected negatively by changes to the code.

3. Illustrative example

A major challenge in RL is exploration [20,21]. Using Scilab-RL, we provide an example workflow on how to modify an existing state-of-the-art RL algorithm (Soft Actor-Critic, SAC [22]) to enhance its capability of exploration. A key feature of SAC is that it incorporates the entropy, or roughly speaking the uncertainty of the agent's policy (actor) as regularization for the optimization problem. This directly motivates our hypothesis that we want to test in the following use case. Not just the uncertainty of the policy but also the uncertainty of the Q function could be used to improve the learning behavior of an agent. A high variance of an ensemble of critics (Q-values) could imply that the agent is uncertain about its current situation and is exploring new state–action pairings in the environment. We assume that encouraging higher variances of critics improves exploration. A straightforward way to test this hypothesis is to use the variance of N critics as an additional intrinsic reward r_i (Eq. (1)), where the variance $\mathrm{Var}[\cdot]$ is defined as the average of the squared deviations from the mean.

$$r_i = \mathrm{Var}[Q_{\Phi 1}(s,a), Q_{\Phi 2}(s,a), \dots Q_{\Phi N}(s,a)] \tag{1}$$

We perform the following steps: First, we clone the Scilab-RL repository. Within the `src/custom_algorithms` folder, we create a copy of the SAC algorithm implementation of Stable Baselines 3 and rename every `sac` to a custom name, e.g. `sac_var`. In addition, we create a respective algorithm configuration file `conf/algorithm/sac_var.yaml` for the hydra hyperparameter management system.

Now, we would be able to use the new (copied) algorithm from the command line via:

```
python src/main.py algorithm=sac_var
env=FetchPush-v1}
```

Next, we modify the algorithm to add the intrinsic rewards. Within the created `sac_var.py` file in the `SAC_VAR` class, we change the `train()` method so that the variance of the critics is computed. Here we add it as intrinsic reward r_i (Eq. (1)) to the extrinsic reward r_e, using a weight η according to Eq. (2). This modified reward is then used to compute the target Q-values with the common Bellman updates. An example algorithm modification is provided as a code excerpt in Appendix B.1.

$$r = (1-\eta) r_e + \eta r_i \tag{2}$$

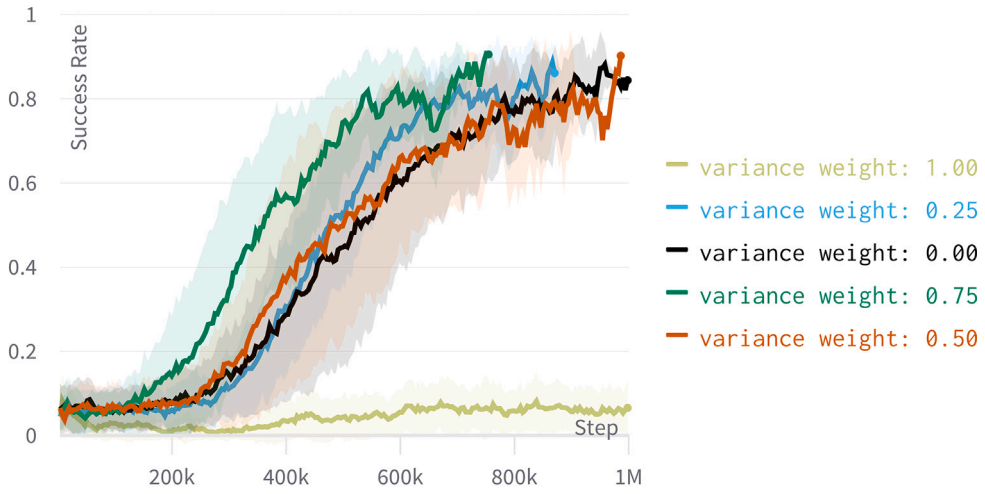


Fig. 3. Hyperparameter optimization for the modified SAC algorithm adding critic variance to the reward. Experiment for the FetchPush environment. Results are improved by adding the variance of the critics as intrinsic reward.

After implementing the novel intrinsic reward, it is desirable to verify that the implementation operates as intended. We can do this verification with the online metric visualization, as depicted in Fig. 2. It shows the rendered visualization of the environment and a corresponding user-defined metric value, in this case the variance of the critics, side by side. In this depicted scenario, the robotic arm accidentally pushes the block (highlighted red) over the edge of the platform, causing it to fall to the ground. As these kinds of situations are rather uncommon, the critics cannot reliably predict, hence we would expect their variance to be high. This is confirmed by the large increase in variance (right of Fig. 2), and indicates that our implementation indeed works as intended. In this regard, online visualization helps to examine the hypothesis made at the beginning of this example that critics' uncertainty may be an indicator of unusual situations and that rewarding uncertainty may encourage exploration.

Having verified that our implementation works as intended, we perform hyperparameter optimization. Here, we optimize the parameter `weight_critic_var` (η in Eq. (2)) for which we want to choose values categorically from [0.0, 0.25, 0.5, 0.75, 1.0]. An example code excerpt for the hyperparameter optimization setting is provided in Appendix B.2. The optimization maximizes the agent's `success_rate`. The `success_rate` describes the proportion of episodes in which the agent terminates in the desired goal state, i.e. was successful [11]. Fig. 3 shows how one can track information on the training progress and evaluation metrics live via the linkage to Weights & Biases. For this example, `weight_critic_var = 0.75` leads to the best results. The graph directly verifies our hypothesis that adding the variance of the critics as an intrinsic reward indeed improves the learning performance. The entire process from setting up the experiment to obtaining results did not require much effort overall using Scilab-RL.

4. Impact

In cognitive modeling and reinforcement learning, setting up an appropriate computational framework for experiments can be a tedious and time-consuming task. Research in RL is quite a matter of trial and error, and the frequency of trying new ideas and testing hypotheses should ideally be high. In addition, it is important to create easier access for non-ML experts and reduce their programming overhead, as the field of AI is quite interdisciplinary and can benefit from a variety of fields such as psychology and cognitive science.

Scilab-RL helps researchers to get started with goal-conditioned RL for robotic agents, and lets them focus on efficient experimenting. The framework is tailored towards the rapid prototyping, development and evaluation of new RL algorithms, methods and environments. This is achieved by bringing together various state-of-the-art RL algorithms, environments, the possibility for hyperparameter optimization and a built-in data visualization for fast and efficient debugging and evaluation. The setup effort is reduced to a minimum so that first agents can be trained right away. The framework is command line based, while it provides a systematic management of configurations for different experiments. Approaches are directly implemented in the cloned repository. Scilab-RL is particularly intended for researchers new to the field, but experts can also benefit as the underlying algorithms and tools are state-of-the-art and constantly updated.

5. Conclusions

We present Scilab-RL, a reinforcement learning framework that accelerates simple and user-friendly experimenting with a minimum of coding effort and setup required. In this paper, we introduced the key features and architecture of the framework and demonstrated its potential usage by means of a typical illustrative example arising from a relevant research question.

Scilab-RL is open for contributions and encourages users to integrate established approaches and algorithms or new challenging environments into the framework and thus make them usable for others.

CRediT authorship contribution statement

Jan Benad: Software, Visualization, Writing – original draft, Writing – review & editing. **Frank Röder:** Software, Writing – original draft, Writing – review & editing. **Manfred Eppe:** Conceptualization, Funding acquisition, Project administration, Software, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors gratefully acknowledge funding by the German Research Foundation DFG through the MoReSpace (402776968) and LeCAREbot (433323019) projects. Additionally, we would like to thank Thilo Fryen and Matin Urdu for their contributions to the codebase.

Appendix A. Intuition for (goal-conditioned) RL

In the following, we provide a brief overview of reinforcement learning (RL) and the technical details used in this article. RL, as a machine learning paradigm, enables agents to learn from scalar feedback, the so-called reward [23]. The agent's objective is to learn a policy that maximizes the cumulative reward over time. For instance, consider the game Tetris where the player needs to rotate shapes falling from the top of the screen depending on the game state in order to maximize the score. The left and right rotation define the available actions to the agent, while the current position of the pieces define the game state. A predefined transition dynamics function determines the logic of the physics involved, hence how the piece interact with the walls and other pieces. A mathematical framework to precisely formulate such a settings is the Markov decision process [24].

Another approach to RL is the concept goal-conditioning. Instead of maximizing a static task-specific reward definition, we consider a dynamic component, the goal, that alters the current reward definition and the required associated actions involved [25]. The core idea is that an agent that can already solve a variety of goals also generalize to solve any other goal [10]. In the goal-conditioned setting the reward signal is sparse. That means that the agent only receives a reward if the whole task is solved instead of after each time step. This yields a harder learning setting but also improves the possibilities of solutions the agent can learn. An example is the game of chess, where the player receives the rewarding feedback only at the end of the game. Because it is very unlikely to win the game by chance and most of the time the player will not get a positive feedback the idea of hindsight experience replay (HER) [11] allows to learn from any type of outcomes by simply assuming in hindsight that it was the intended goal configuration.

Appendix B. Code excerpts for the illustrative example

B.1. Modified SAC algorithm

```
class SAC_VAR(OffPolicyAlgorithm):
  def __init__(
    self,
    ... # left out for clarity
    weight_critic_var: float,

  ):
    super(SAC_VAR, self).__init__(
      ... # left out for clarity
    )
    self.weight_critic_var = weight_critic_var

  def train(
    self,
    gradient_steps: int,
    batch_size: int = 64
  ) -> None:
    # compute the variance of Q values
    q_values = th.cat(
      self.critic_target(
       replay_data.next_observations, next_actions
      ),
      dim=1
    )
    critic_variance = q_values.var(dim=1)
    # min-max scale variance between 0 and 1
    var_max = th.max(critic_variance)
    var_min = th.min(critic_variance)
    critic_variance = (
     (critic_variance - var_min) / (var_max - var_min)
    )
    critic_variance = critic_variance.unsqueeze(1)
    critic_variances.append
    (critic_variance.tolist())
    # extrinsic + intrinsic reward (weighted)
    reward_mod = (
      (1 - self.weight_critic_var)
      * replay_data.rewards \
      + self.weight_critic_var * critic_variance
    )
    # compute target Q-values
    target_q_values = (
      reward_mod + (1 - replay_data.dones) \
      * self.gamma * next_q_values
    )
    ... # left out for clarity
```

B.2. Hydra sweeper configuration

```
env: 'FetchPush-v1'
hydra:
 sweeper:
  study_name: sac_var_FetchPush
  max_trials: 64
  n_jobs: 16
  direction: maximize
  min_trials_per_param: 3
  max_trials_per_param: 12
  search_space:
   ++algorithm.weight_critic_var:
    type: categorical
    choices:
     - 0.0
     - 0.25
     - 0.5
     - 0.75
     - 1.0
```

References

[1] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of go with deep neural networks and tree search. Nature 2016;529(7587):484–9. http://dx.doi.org/10.1038/nature16961.

[2] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. Playing atari with deep reinforcement learning. 2013, http://dx.doi.org/10.48550/arXiv.1312.5602, arXiv:1312.5602.

[3] Fawzi A, Balog M, Huang A, Hubert T, Romera-Paredes B, Barekatain M, Novikov A, R. Ruiz FJ, Schrittwieser J, Swirszcz G, Silver D, Hassabis D, Kohli P. Discovering faster matrix multiplication algorithms with reinforcement learning. Nature 2022;610(7930):47–53. http://dx.doi.org/10.1038/s41586-022-05172-4.

[4] Achiam J. Spinning up in deep reinforcement learning. 2018.

[5] Huang S, Dossa RFJ, Ye C, Braga J, Chakraborty D, Mehta K, Araújo JaGM. CleanRL: high-quality single-file implementations of deep reinforcement learning algorithms. J Mach Learn Res 2022;23(274):1–18.

[6] garage contributors T. Garage: A toolkit for reproducible reinforcement learning research. GitHub Repos 2019.

[7] Liang E, Liaw R, Nishihara R, Moritz P, Fox R, Goldberg K, Gonzalez J, Jordan M, Stoica I. RLlib: Abstractions for distributed reinforcement learning. In: Proceedings of the 35th International Conference on Machine Learning. PMLR; 2018, p. 3053–62.

[8] Weng J, Chen H, Yan D, You K, Duburcq A, Zhang M, Su Y, Su H, Zhu J. Tianshou: A highly modularized deep reinforcement learning library. J Mach Learn Res 2022;23(267):1–6.

[9] Raffin A. RL Baselines3 Zoo. GitHub Repos 2020.

[10] Schaul T, Horgan D, Gregor K, Silver D. Universal value function approximators. In: Proceedings of the 32nd International Conference on Machine Learning. PMLR; 2015, p. 1312–20.

[11] Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, McGrew B, Tobin J, Pieter Abbeel O, Zaremba W. Hindsight experience replay. In: Advances in Neural Information Processing Systems. 30, Curran Associates, Inc.; 2017.
[12] Towers M, Kwiatkowski A, Terry J, Balis JU, De Cola G, Deleu T, Goulão M, Kallinteris A, Krimmel M, KG A, et al. Gymnasium: A standard interface for reinforcement learning environments. 2024, arXiv preprint arXiv:2407.17032.
[13] Todorov E, Erez T, Tassa Y. MuJoCo: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2012, p. 5026–33. http://dx.doi.org/10.1109/IROS.2012.6386109.
[14] Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19, New York, NY, USA: Association for Computing Machinery; 2019, p. 2623–31. http://dx.doi.org/10.1145/3292500.3330701.
[15] Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N. Stable-Baselines3: reliable reinforcement learning implementations. J Mach Learn Res 2021;22(268):1–8.
[16] Zaharia M, Chen A, Davidson A, Ghodsi A, Hong S, Konwinski A, Murching S, Nykodym T, Ogilvie P, Parkhe M, Xie F, Zumar C. Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng Bull 2018;41:39–45.
[17] Biewald L. Experiment tracking with weights and biases. 2020.
[18] de Lazcano R, Andreas K, Tai JJ, Lee SR, Terry J. Gymnasium robotics. 2024.
[19] Yadan O. Hydra - A framework for elegantly configuring complex applications. 2019.
[20] Pathak D, Agrawal P, Efros AA, Darrell T. Curiosity-driven exploration by self-supervised prediction. In: Proceedings of the 34th International Conference on Machine Learning. PMLR; 2017, p. 2778–87.
[21] Burda Y, Edwards H, Storkey A, Klimov O. Exploration by random network distillation. 2018, arXiv:1810.12894.
[22] Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning. PMLR; 2018, p. 1861–70.
[23] Sutton RS, Barto AG. Reinforcement learning: An introduction, In: A bradford book, second ed.. MIT Press; 2018-11.
[24] Bellman R. A Markovian decision process. J Math Mech 1957;6(5):679–84.
[25] Kaelbling LP. Learning to achieve goals. In: International Joint Conference on Artificial Intelligence. Morgan Kaufmann; 1993, p. 1094–8.