Crux.jl: Deep Reinforcement Learning in Julia

Robert J. Moss [1], **Anthony Corso** [1], **and Mykel J. Kochenderfer** [1]

1 Stanford University

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @DennisWayo
- @Luis-Varona

Submitted: 25 September 2025
Published: unpublished

Summary

Crux.jl is a Julia library for deep reinforcement learning (RL) that provides concise, modular implementations of widely used algorithms. The package offers CPU/GPU-accelerated training using Flux.jl (Innes, 2018) and is built upon shared abstractions (policies, value functions, buffers, objectives, and update rules). These abstractions helps with both code reuse and understanding the core differences between algorithms (e.g., their surrogate losses, trust-region constraints, or advantage estimations). Crux.jl includes policy-gradient and actor-critic methods such as REINFORCE (Williams, 1992), PPO (Schulman et al., 2017), and TRPO (Schulman et al., 2015), along with off-policy value-based and actor–critic variants such as DQN (Mnih et al., 2015), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018), with additional support for imitation, offline, adversarial, and continual learning algorithms. Shown in fig. 1, the library integrates with POMDPs.jl (Egorov et al., 2017) and the Python gymnasium environments (Towers et al., 2024) for reproducible benchmarking and fast experimentation.

```julia
using Crux, POMDPGym

problem = GymPOMDP(:CartPole)
as = actions(problem)
S = state_space(problem)

# Flux actor and critic networks
A() = DiscreteNetwork(Chain(Dense(dim(S)..., 64, relu), Dense(64, length(as))), as)
V() = ContinuousNetwork(Chain(Dense(dim(S)..., 64, relu), Dense(64, 1)))

# Setup solvers and solve to get their respective policies
solver_reinforce = REINFORCE(S=S, π=A())
policy_reinforce = solve(solver_reinforce, problem)

solver_a2c = A2C(S=S, π=ActorCritic(A(), V()))
policy_a2c = solve(solver_a2c, problem)

solver_ppo = PPO(S=S, π=ActorCritic(A(), V()))
policy_ppo = solve(solver_ppo, problem)
```

Figure 1: Crux.jl usage for the cart-pole problem, solved using various deep RL algorithms.

Statement of Need

Reinforcement learning libraries, such as Stable Baselines3 (Raffin et al., 2021) and RLlib (Liang et al., 2018), often blur the distinction between algorithmic ideas and framework code, hindering fair comparison, reuse, and extension to settings such as partial observability, safety constraints, or offline data. Crux.jl is a compact, Julia-native framework built on multiple dispatch and Flux.jl that factors training into explicit, swappable components such as policies, critics, buffers, return/advantage estimators, objectives, and update rules. In contrast to the inheritance-based code for Stable Baseline3, Crux.jl implements the `DQN` solver using a simple `dqn_target` function for the `OffPolicySolver` type, and a separate `td3_target` function for the same `OffPolicySolver` when implementing `TD3`. This design enables rigorous, reproducible experimentation across RL settings, and integration with POMDPs.jl and gymnasium standardizes environment interaction and evaluation. In short, Crux.jl provides a principled, composable deep RL framework for Julia that enables rapid ablations, fair baselines, and reproducible results without sacrificing performance or clarity.

Research and Industrial Usage

The design goals of Crux.jl are reflected in its use across a range of scientific and applied domains. In aerospace, this package has been applied to energy-optimized path planning for unmanned aircraft in varying winds (Banerjee & Bradner, 2024). In computational physics, researchers have combined reinforcement learning with metaheuristics for Feynman integral reduction, demonstrating the method's role in symbolic and high-performance computing processes (Zeng, 2025). More broadly, Crux.jl has been used to prototype algorithms for validation of safety-critical systems (Kochenderfer et al., 2026), where component-wise modularity and reproducibility are particularly valuable.

Acknowledgments

The authors would like to thank the Stanford Center for AI Safety for funding this work.

References

Banerjee, P., & Bradner, K. (2024). Energy-optimized path planning for UAS in varying winds via reinforcement learning. *AIAA AVIATION FORUM and ASCEND*. https://doi.org/10.2514/6.2024-4545

Egorov, M., Sunberg, Z. N., Balaban, E., Wheeler, T. A., Gupta, J. K., & Kochenderfer, M. J. (2017). POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, *18*(26), 1–5.

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning (ICML)*, 1587–1596.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning (ICML)*, 1861–1870.

Innes, M. (2018). Flux: Elegant machine learning with Julia. *Journal of Open Source Software*, *3*(25), 602. https://doi.org/10.21105/joss.00602

Kochenderfer, M. J., Katz, S. M., Corso, A. L., & Moss, R. J. (2026). *Algorithms for Validation*. MIT Press.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., & Ion Stoica. (2018). RLlib: Abstractions for distributed reinforcement learning. *International Conference on Machine Learning (ICML)*.

Mnih, V., Kavukcuoglu, K., Silver, D., & others. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

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

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International Conference on Machine Learning (ICML)*, 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv:1707.06347*. https://doi.org/10.48550/arXiv.1707.06347

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., & others. (2024). Gymnasium: A standard interface for reinforcement learning environments. *arXiv:2407.17032*. https://doi.org/10.48550/arXiv.2407.17032

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256. https://doi.org/10.1007/BF00992696

Zeng, M. (2025). Reinforcement learning and metaheuristics for Feynman integral reduction. *arXiv:2504.16045*. https://doi.org/10.48550/arXiv.2504.16045