PufferLib 2.0: Reinforcement Learning at 1M steps/s

Joseph Suarez

Keywords: PufferLib, Reinforcement Learning, Library, Tools

Summary

PufferLib is an open-source reinforcement learning project built around efficient and broadly compatible simulation. Our first-party suite of 12 environments each run at 1M steps/second. For existing environments, PufferLib provides one-line wrappers that eliminate common compatibility problems and fast vectorization to accelerate training. With PufferLib, you can use familiar libraries like CleanRL and SB3 to scale from classic benchmarks like Atari and Procgen to complex simulators like NetHack and Neural MMO 3. Code, documentation, demos, and less formal blog coverage are available at puffer.ai.

Contribution(s)

1. One-line wrappers that make complex environments like Nethack, Neural MMO, Griddly, etc. compatible with any RL library that supports standard Gymnasium/PettingZoo formats.
   Context: Gymnasium and PettingZoo are the most widely used environment formats. This means PufferLib is compatible with the vast majority of environments using only a 1-line wrapper.
2. Drop-in vectorization for simulating environments in parallel. Most environments will see at least a 30% speed boost and 50%-3x with pooling. This is a broadly compatible contribution applicable to nearly all environments.
   Context: Gymnasium provides the most common vectorization backend. It is slow for the reasons outlined in the paper.
3. Puffer Ocean, a suite of 12 environments written in C that each simulate at >1M steps/second on a single CPU core.
   Context: A few of these have built-in AI opponents that can slow performance when search depth is increased. Base speed is >1M steps/second on a high end desktop core. Some environments run 10M steps/second.
4. A PPO demo that trains Ocean environments at 300k-1.2M steps/second on a single RTX 4090. Our standard architectures are MLP-LSTM or CNN-LSTM from 150k-1M parameters.
   Context: It's compatible with all third-party environments. Training is up to 30k steps/second on Atari, which is still 30x faster than the original CleanRL.

PufferLib 2.0: Reinforcement Learning at 1M steps/s

Joseph Suarez
jsuarez@puffer.ai

PufferAI

Abstract

PufferLib is an open-source reinforcement learning project built around efficient and broadly compatible simulation. Our first-party suite of 12 environments each run at 1M steps/second. For existing environments, PufferLib provides one-line wrappers that eliminate common compatibility problems and fast vectorization to accelerate training. With PufferLib, you can use familiar libraries like CleanRL and SB3 to scale from classic benchmarks like Atari and Procgen to complex simulators like NetHack and Neural MMO 3.

Versioning Note

We developed and released PufferLib 3.0 during the review cycle for RLC. The latest version includes core algorithmic improvements over PPO and a new hyperparameter tuning algorithm. Ocean has grown to over 25 environments, and small models now train 3-5M steps/second. Our baselines solve Ocean Pong in 3-5 seconds, Breakout in 20-30 seconds, and most other environments in 1-10 minutes on a single RTX 5090. The largest experiment so far on 3.0 trains Neural MMO 3 for 640B steps in 3 days and 8 hours on 6 RTX 4090s. This is over a petabyte of observations, or roughly 12,000 years of games. All of these changes are too material to introduce in this manuscript after acceptance, but they are covered by a series of blogs on puffer.ai and 3.0 is fully open source.

1 Background and Introduction

Reinforcement learning research is slow and cumbersome. Atari games have been the most widely used benchmark since the introduction of the Arcade Learning Environment (Bellemare et al., 2012) in 2012. These are single-agent tasks that run at a few thousand steps per second on a modern CPU core. Standard learning libraries like CleanRL and SB3 then introduce old, unoptimized multiprocessing and expensive Python environment wrappers. The result is training that runs at hundreds to thousands of steps per second, leaving modern GPUs at 5-20% utilization. This is particularly limiting for RL research given the common sensitivity to hyperparameters. Accurately comparing methods often requires hundreds or thousands of experiments. This is impractical without large-scale industry resources.

The problem quickly becomes much worse on more complex environments. Gym/Gymnasium do not have multi-agent support natively. Every single work claiming to have a multi-agent Gym interface is hacking the API slightly differently from every other project. PettingZoo provides native multi-agent support, but it does not have native multiprocessing support for parallel simulation, and the API itself introduces heavy overhead. Given that most new environments are written by researchers in Python, they are not particularly fast to begin with. All of this makes doing RL research on progressively more interesting tasks slow and cumbersome.

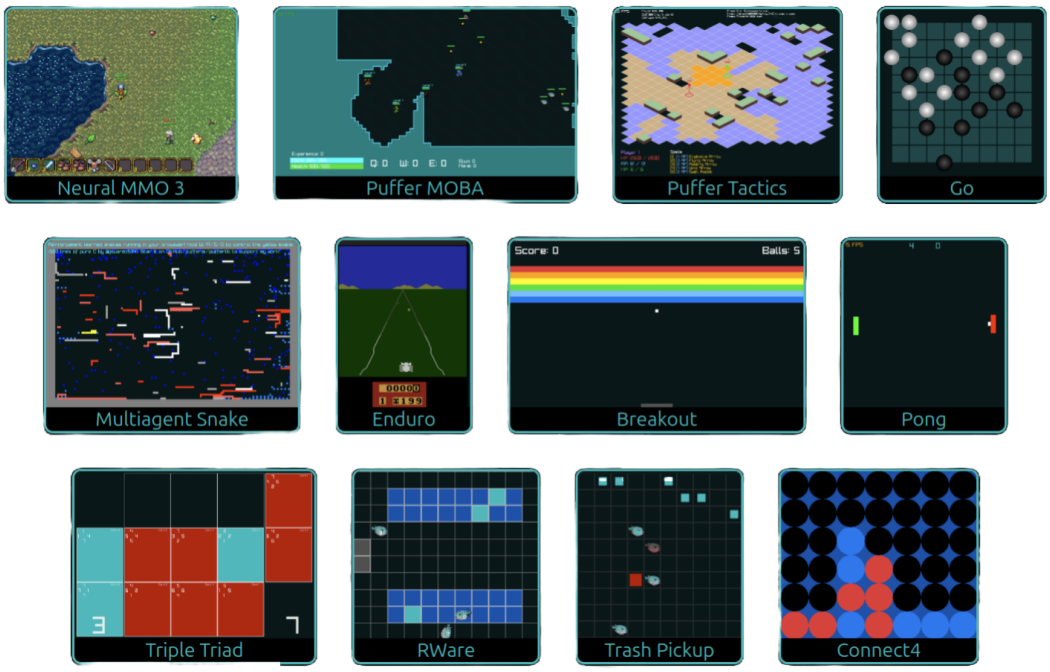

Figure 1: Ocean is PufferLib's suite of first-party environments. There are currently 12 environments that run 1M+ steps/second on a single CPU core. They range from as simple as Pong to as complex as Neural MMO 3.

Gym/Gymnasium (Brockman et al., 2016; Towers et al., 2023) These libraries define the API for the vast majority of environments used in research and provide associated tools, such as the `spaces` module for defining data shapes and `vector` for vectorization. An important piece of historical context: OpenAI originally published Gym in 2016, and nearly all non-DeepMind environments from 2016-2022 used it. Maintenance was taken over by an independent open source group in 2021, but since they did not have full control of the repository required for maintenance, they forked it as Gymnasium in 2022. The Gym library is no longer maintained.

PettingZoo: (Terry et al., 2020) A multiagent analog to Gym/Gymnasium. It aims to support very complex environments, including those with heterogeneous observation and action spaces, but some of the surrounding tooling suffers in performance as a result.

DM Env: (Muldal et al., 2019) DeepMind's counterpart to Gym. Both provide comparable APIs, and there is no real reason to use one over the other, but most modern research has standardized on Gymnasium. PufferLib converts to the Gymnasium API by default for this reason.

PufferLib is a library of environments and broadly-compatible tools for fast and simple reinforcement learning. In contrast, the main purpose of the libraries above is to define standards on how environments should be implemented. They are compatible with but not directly comparable to PufferLib.

Compared to other reinforcement learning libraries, PufferLib does not aim to be a collection of algorithm implementations. You can integrate our tools with the library of your choice. Our main demo is built on CleanRL, but with added integrations for performance, logging, experiment management, and hyperparameter sweeps. We include a list of related libraries that may include some utilities, though they primarily target learning itself.

1. **CleanRL:** (Huang et al., 2021) Single-file implementations with research-friendly features.
2. **Stable Baselines 3 (SB3):** (Raffin et al., 2021) Reliable implementations for a wide range of algorithms.
3. **Sample Factory 2.0:** (Petrenko et al., 2020) High-throughput reinforcement learning with a smaller selection of algorithms
4. **Tianshou:** (Weng et al., 2022a) Another widely used RL library. Comparable in design to SB3 with similar tradeoffs.
5. **TorchRL:** (Bou et al., 2023) Official PyTorch RL library. This is a relatively newer project.

Note that this list omits libraries focused on control and robotics environments, most of which are simulated on the GPU or are in the process of being ported to the GPU. PufferLib currently focuses on tooling for CPU environments. Only certain types of environments, such as those with heavy physics, are sensible to implement on GPU. As a result, we consider these parallel areas of tooling, each of which is unlikely to replace the other. Recently, we have included bindings for a few GPU environments, though these are mainly for groups that have wanted to use our demo script for its speed, logging, and hyperparameter tuning utilities.

2 PufferTank

Problem: Setting up a development environment for RL is difficult and time consuming. Extensive system package dependencies, slow source builds, and versioning obstacles prevent otherwise useful environments from attaining broader adoption.

Solution: PufferTank is a Docker container with dependencies set up correctly and tricky versioning issues handled. The current version has CUDA 12.1, PyTorch 2.4, Python 3.11, dozens of environments, and convenient development features such as rendering settings for Linux and WSL. We are aware that containerized development is not ubiquitous in AI and thoroughly considered lighter weight options. Unfortunately, the dependencies for many third-party environments take hours to build and contain a mix of system and python packages. Our prebuilt images allow you to get up and running in minutes and to reset to the base installation instantly if anything goes wrong. Most IDEs, including VSCode, have plugins for seamless development in both local and remote containers. For minimalists, NeoVim with SuperMaven code completion is preinstalled.

Note that **you can use PufferLib without PufferTank**. PufferLib itself is a PyPI package – PufferTank exists because *specific common environments* have tricky dependencies, not PufferLib itself.

3 PufferLib

3.1 Ocean

Problem: The vast majority of reinforcement learning environments are 10-100x too slow to keep up with an optimized GPU trainer, even after heavy optimization. This problem will only get worse in the future, as GPU performance is improving faster than CPU performance. Using only GPU-accelerated environments is not realistic for reasons already discussed.

Solution: PufferLib ships 20,000 lines worth of C environments. These are far easier to develop than high-performance GPU environments. The proof of this is that around half of this code was written by independent open-source contributors, many of whom had no prior RL experience. We effectively turned high-performance environment creation into low-level game development. Another 6,000 lines are currently under review. The current list of environments includes **pong**, **breakout**, and **enduro** implementations as analogs to the standard Atari tasks, **Connect4**, **TripleTriad**, and **Go** as board/card games, **RWare** and **TrashPickup** as grid-world pick-and-place tasks, **Snake** for multi-agent research, and three scale-up tasks:

1. **Neural MMO 3:** A massively multiagent game environment featuring hundreds to thousands of agents, procedural map generation, foraging, combat, leveling, items, equipment, and trade.
2. **MOBA:** A miniature version of DoTA, the 5v5 esport tackled by OpenAI Five. Our version has 5 heros with 3 unique abilities each as well as lane creeps, neutrals, and tower placement mostly consistent with the original game.
3. **Tactics:** A CRPG-style 2d combat simulator based on games like Fire Emblem and Dofus. This is the only environment in our suite for which we do not yet have a benchmark – we will be done with this tuning soon.

A key innovation of PufferLib is that all of these environments can simulate their observations directly into shared memory. This eliminates several redundant data copies. To our knowledge, these are present in every other RL library and environment. Puffer Ocean also provides a small suite of sanity environments designed to catch common RL implementation errors:

- **Squared:** Agent starts at the center of a square grid. Targets are placed on the perimeter of the grid. Reward is 1 minus the L-inf distance to the closest target. This means that reward varies from -1 to 1. Reward is not given for targets that have already been hit.
- **Password:** Guess the password, which is a static binary string. The policy has to not determinize before it happens to get the reward, and it also has to latch onto the reward within a few instances of getting it.
- **Stochastic:** The optimal policy is to play action 0 p percent of the time and action 1 $(1-p)$ percent of the time. This is a test of whether the algorithm can learn a nonuniform stochastic policy.
- **Memory:** Repeat the observed sequence after a delay. It is randomly generated upon every reset. The sequence is presented one digit at a time, followed by a string of 0.
- **Multiagent:** Agent 1 must pick action 0 and Agent 2 must pick action 1.
- **Spaces:** A simple environment with hierarchical observation and action spaces. Obtaining maximal score requires taking into account all subspaces.
- **Bandit:** Simulates a classic multiarmed bandit problem.

3.2 Emulation

Problem: Learning libraries make strong assumptions that are incompatible with cognitively interesting and efficient environments. This is partially a historical problem and partially an engineering limitation. Most learning libraries were designed with Atari in mind, which is single-agent with flat tensor (image) observations and single discrete actions. The most interesting environments usually do not adhere to these constraints. For example, NetHack is rendered in ASCII with additional information contained in a panel at the bottom of the console. There is no sensible way to represent this information as a flat tensor, so the NetHack Learning Environment (Küttler et al., 2020) is forced to expose it as a dictionary of arrays with different shapes. Even when flat representations are possible, they are not always optimal. For example, Neural MMO (Suarez et al., 2021) provides isometric rendering, but the training API instead exposes local state data for each of a variable number of agents because it is 1000x faster, enabling users to train on a desktop instead of a supercomputer.

Solution: PufferLib provides one-line wrappers that make simple learning libraries work with complex environments. It does so by flattening observations to tensors and actions to a single multidiscrete variable. This means that, from the perspective of the learning library, the environment *looks* like Atari, thereby *emulating* a simpler environment. PufferLib provides a function to undo this operation, which you can call in the first line of your model's forward pass. This means that there is **no loss of generality**.

Normally, to use an environment like NetHack, you would first have to write a single-purpose environment wrapper to do something similar to what PufferLib does more generally. Since this requires packing and unpacking arbitrary data, it is an error prone operation that is difficult to test. PufferLib

does all of this for you with an efficient implementation tested against dozens of real and mocked environments.

PufferLib's approach is so simple as to seem obvious, but there are many seemingly reasonable ways to accomplish the same task that do not work. For example, several reinforcement learning libraries attempt to natively support hierarchical observation and action data with no flattening. This dramatically complexifies the code base and also prevents multiple important optimizations later in the data pipeline, most notably during vectorization. Emulation overhead is negligible for environments slower than several thousand steps per second per core.

PufferLib's emulation layer also handles other niche but important compatibility issues. For example, it will perform shape checks on the first batch of data. This catches nearly all user errors but does not add any overhead, since the checks are only performed at startup. In multiagent environments, PufferLib ensures that observations and actions are returned in a canonical sorted order. If the environment has a variable number of agents, PufferLib will pad observations to maintain fixed size data buffers. These are all common sources of difficult to diagnose bugs.

3.3 Environments

PufferLib currently provides bindings for atari, procgen, nethack, neural mmo, minigrid, minihack, crafter, griddly, pokemon, and more. These are not the only supported environments, just the ones that we have manually tested for dependency and versioning quirks.

Problem: Many widely used environments have unresolved versioning and API compatibility issues with Gym/Gymnasium. This is in addition to the system dependencies resolved by PufferTank.

Solution: PufferLib provides known-good bindings for dozens of popular environments. These include Gym/Gymnasium conversion, the standard emulation wrapper, and sometimes additional fixes for specific quirks. There are installation options for each package, e.g. `pip install pufferlib[atari]` or `pip install pufferlib[nethack]`. PufferLib provides additional insurance against poor dependency management – for example, we pin versions for sub-packages that are know to commonly ship breaking changes. There is also a `pufferlib[common]` option, which installs the broadest set of mutually compatible environments. This is included with PufferTank by default, so most users should have an out-of-the-box development experience. It was also one of the motivations for building PufferTank in the first place, since some of the `common` environments require additional system packages. Note that **pufferlib does not have a registry** by design, so there are **no additional requirements for custom environments**.

3.4 Vectorization

Vectorization is the process of simulating many environments, usually on different cores, which requires aggregating observations and distributing actions. Learning libraries either use Gym/Gymnasium's builtin vectorization or ship their own.

Problem: Existing vectorization methods are slow and provide limited or no support for complex environments with e.g. structured observations, multiple agents, variable population size, etc.

Solution: PufferLib implements fast and broadly compatible vectorization from scratch. We provide serial, multiprocessing, and Ray backends with the same API. The greatest focus is on the multiprocessing backend. **All of the above combined are implemented in only a few hundred lines.** For additional details, the code is truly quite simple to read.

A hard assumption on PufferLib emulation. Earlier, we said that other libraries attempting to implement native structured data processing would cause issues. This is one such instance. Gymnasium and SB3, the two most popular vectorization implementations, both attempt native support. The Gymnasium implementation misses several crucial opportunities for optimization as a result. The SB3 implementation simply flattens observations, without giving the user any way to unflatten them.

For some reason, it does this on the main process and with a rather inefficient implementation. By comparison, PufferLib's implementation is shorter, faster, and more flexible. The features described below would be quite difficult to implement otherwise.

Native multiagent support. Most vectorization implementations, including PettingZoo's, bolt this on with unoptimized wrappers. Both SB3 and Gymnasium have made clear that there will never be official multiagent support.

A Python implementation of EnvPool (Weng et al., 2022b). Standard vectorization simulates M environments in parallel and requires waiting on all M before returning observations. PufferLib can instead retrieve $N << M$ observations. This has two important implications. First, by setting $M = 2N$, simulation becomes approximately double-buffered. This means that CPU cores are processing half of the environments while the GPU computes actions for the other half. Second, by setting $M \gg 2N$, the model no longer has to wait on the slowest environments before returning a batch of observations. This feature is especially important in complex environments, which tend to have more branching logic paths. At time of writing, this is the only existing Python implementation of EnvPool, as opposed to the original EnvPool implementation that supports select C++ environments.

Multiple environments per worker. PufferLib allows you to specify the number of environments and the number of workers separately. When a worker is responsible for multiple environments, it efficiently stacks data returned by each sub-environment in preallocated arrays without performing any extra copies. This feature is important when running many more environments than your machine has cores, as it avoids clogging the system with small processes.

Shared memory for data communication. We load `observations`, `rewards`, `terminals`, `truncations`, and `actions` signals into large shared arrays. We use pipes, which are up to 10x faster than queues due to a Python quirk, for communicating `infos`. Empty `infos` are pruned, and we provide wrappers to aggregate them over episodes. As a result, only one step per episode requires any inter-process communication. By comparison, SB3 does not have shared memory. Gymnasium provides a slower shared memory implementation that attempts to handle structured data natively, requiring multiple small copy operations and additional Python logic.

Shared flags for signaling. Worker processes busy-wait on an unlocked shared array flag to detect when actions are ready and update the flag after computing observations. This almost completely eliminates inter-process communication overhead. Pipes are only used when an environment returns non-empty `infos`, which will be once per episode when using our wrappers.

Zero-copy batching. For environments with large observations, we provide a setting to load batches of data directly from shared memory by waiting on a contiguous subset of worker process indices. Other settings require one copy. A naive implementation not using shared memory would require two or three copies.

Four separately optimized code paths. For fast environments, main process overhead has to be optimized to within a few microseconds. Even operations like manipulating process IDs in a list can result in noticeable performance drops. We identify and separately optimize four common workload cases. In the synchronous case, environments are split evenly across cores and loading into a single batch in shared memory with no extra copy operations. In the fully asynchronous case, data is taken from the first workers to finish, requiring a single copy operation to load the batch into contiguous memory. There is a special case of the latter where each batch is simulated on a single worker, so it can be loaded without additional copies. There is also the above zero-copy case, which is roughly equivalent to a circular buffer of batches.

An autotune utility. Obtaining the best configuration for your environment and hardware requires testing all four code paths. We provide an utility that benchmarks valid vectorization settings.

Environment	SPS	% Reset	% Step STD	% Overhead
Neural MMO 2	2400	68	59	7.2
Nethack	39k	0.63	45.3	64
MiniHack	11k	2.1	28	4.9
Pokemon Red	700	0.00	43	0.08
Cartpole	270k	18	37	13
Ocean Squared	240k	55	53	14
Procgen Bigfish	25k	0.36	14	2.5
Atari Breakout	1.2k	54	4.3	0.16
Crafter	320	80	26	0.04
Minigrid	16k	4.5	8.1	2.7

Table 1: Updated single-core throughput and emulation overhead of various environments, evaluated on the desktop. Steps per second (SPS) is timed with emulation. Some environments have slow resets or high per-step variance.

3.5 Models

PufferLib provides an optional model format that splits the normal PyTorch `forward` function into separate `encode` and `decode` functions. This allows PufferLib to sandwich an LSTM between the computation of hidden state and actions. We apply this operation as a wrapper, meaning that LSTM support becomes optional and configurable per-experiment, without having to write two models. Users are free to disregard this feature and use other frameworks for LSTM support or implement it manually. However, our users have found this feature important, since LSTM state reshaping operations are one of the most common sources of difficult to diagnose bugs.

Pufferlib includes a few baseline models such as the original NatureCNN and ResNet models from DQN (Mnih et al., 2013) Impala (Espeholt et al., 2018), respectively. There is also a default architecture which defines an MLP sized to the flat observation and action spaces. This is useful for small test environments. **All base models directly subclass torch.nn.Module.** There is no additional model layer required by PufferLib.

4 Performance

PufferLib's emulation layer typically adds a few tens of microseconds to simulation time. As shown in Table 1, the overhead is negligible for environments slower than a few thousand steps per second. Emulation works by inferring a numpy structured array datatype from the environment's Gym/Gymnasium observation and action spaces. This is an analog to C structs that provides an efficient numpy interface over structured data in contiguous memory. Conveniently, we can use structured arrays as flat bytes, as is required for efficient vectorization, or with dict-like accessors, as is required by the model and environment. This critical piece of code is Cythonized and tested to be faster than a half dozen implementations from earlier in development, including efforts to write it in C and Rust. Suffice that our implementation is, at the least, probably faster than the naive one-off script that users would have to write for any specific environment otherwise.

PufferLib's vectorization is faster than both the Gymnasium and SB3 implementations in almost all cases, even without our EnvPool feature enabled, which provides most of the speedup. **The following comparisons are a worst case scenario for PufferLib** because they also rely on PufferLib's emulation. The fairer comparison would be to time PufferLib's emulation + vectorization vs. other implementations without PufferLib emulation. This would trigger inefficient code paths, such as Gymnasium's structured shared memory processing and SB3' main-thread flattening. However, several of the test environments simply would not work without PufferLib's emulation. By comparison, we had never explicitly tested any of these environments with Gymnasium/SB3 vectorization, and they all worked on the first try.

Environment	Pufferlib (D/L)	Puffer Pool (D/L)	Gymnasium (D/L)	SB3 (D/L)
Neural MMO	13k / 4.5k	19k / 7.2k	- / -	- / -
Nethack	96k / 18k	- / -	7k / 7k	6k / -
Minihack	55k / 8k	- / -	11k / 7k	12k / 6k
Pokemon Red	5k / 650	7.2k / 830	5k / 960	4.7k / 1k
Cartpole	460k / 110k	3M / 200k	82k / 14k	100k / 11k
Ocean Squared	490k / 110k	4M / 190k	116k / 12k	99k / 12k
Procgen Bigfish	33k / 6.2k	150k / 9.6k	41k / 5k	30k / 4.8k
Atari Breakout	11.8k / 2k	25.6k / 3k	4.8k / 2.5k	3.3k / 2.3k
Crafter	360 / 170	2.8k / 450	340 / 135	350 / 203
Minigrid	151k / 16k	210k / 20k	54k / 10k	44k / 10k

Table 2: Vectorized throughput of PufferLib (with and without EnvPool), SB3, and Gymnasium. Evaluated on the desktop (D) and laptop (L).

We benchmark a variety of real environments with different step times, step time variances, and observation/action spaces. We use two separate machines for testing. The first machine is a commercial desktop with a 24-core i9-14900k processor and an RTX 4090 GPU. The second machine is a laptop with a 6-core i7-10750H processor and an RTX 3070 GPU. Table 2 reports the results of these experiments.

The Gymnasium and SB3 multiprocessing implementations experience significant scaling degradation above 1000 synchronizations per second per core. Instead of clogging the system with small processes, PufferLib provides an optimized implementation for running multiple environments/core. This allows it to scale even to environments that run at 100k+ steps/second.

The other major source of performance improvement is PufferLib's EnvPool implementation. This is another place where our benchmarks are unfair to our own implementation: the tests are run without a model in a loop. A major benefit of EnvPool is allowing the environments to continue computing observations while the policy is computing actions. This can drive GPU idle time to 0.

Even in this worst case scenario, we generally obtain at least a 30-40% performance improvement with EnvPool. There are important cases where the improvement can be much larger. For example, Crafter is 6x faster with Puffer Pool. The reason for this is that Crafter has especially long reset times and high step time variance. You may have also noticed that PufferLib scales better on the desktop (D) than on the laptop (L). The chipset used has 8 *performance cores* and 16 slower *efficiency cores*. This is an increasingly common design in high-end Intel chips. SB3 and Gymansium are bottlenecked by the slowest environment and the slowest core, while PufferLib will retrieve observations from the first environments to finish processing.

5 First-party Training with Clean PuffeRL

By design, PufferLib does not include a library of learning algorithm implementations and we have no plans to develop one. We do maintain one heavily customized version of CleanRL's PPO (Schulman et al., 2017) implementation for testing and baselines. It has been expanded to allow separate training and evaluation, model saving and checkpointing, faster LSTM support, better logging and WandB integration, asynchronous environment simulation, and additional features for multiagent learning. We include a runner file with a CLI for all included PufferLib environments, clean YAML configs, and integration with WandB for tracking, baselines, and hyperparameter tuning.

6 Limitations

PufferLib is missing some of the newer Gymnasium spaces. Few environments use these, and integrations are planned based on user demand. There are a few edge cases in vectorization where

PufferLib's synchronous multiprocessing is slower than Gymnasium/SB3. This is a quirk of OS process switching. We have not received any user reports of particularly egregious cases.

While we provide our integration with Neural MMO 2 as evidence of our library's efficacy in complex multiagent domains, there are simply no good multiagent vectorization implementations to compare against. We fought with deprecated PettingZoo wrappers for a while, but they did not work with Neural MMO. This alone should be a good indicator of PufferLib's impact.

7 Conclusion

PufferLib provides fast and flexible tools for reinforcement learning. The majority of new users will benefit immediately from improved performance and a smoother development experience. However, the real value of PufferLib is the ease with which researchers can move their work to more complex and interesting environments, which have traditionally been difficult to work with. We hope that PufferLib will allow the field to explore new ideas hitherto constrained by the simplicity of readily available environments.

References

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL http://arxiv.org/abs/1207.4708.

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, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *CoRR*, abs/2111.08819, 2021. URL https://arxiv.org/abs/2111.08819.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *CoRR*, abs/2006.13760, 2020. URL https://arxiv.org/abs/2006.13760.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Alistair Muldal, Yotam Doron, John Aslanides, Tim Harley, Tom Ward, and Siqi Liu. dm_env: A python interface for reinforcement learning environments, 2019. URL http://github.com/deepmind/dm_env.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav S. Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 FPS with asynchronous reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 7652–7662. PMLR, 2020. URL http://proceedings.mlr.press/v119/petrenko20a.html.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Joseph Suarez, Yilun Du, Clare Zhu, Igor Mordatch, and Phillip Isola. The neural mmo platform for massively multiagent research. In J. Vanschoren and S. Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/44f683a84163b3523afe57c2e008bc8c-Paper-round1.pdf.

Justin K. Terry, Benjamin Black, Ananth Hari, Luis S. Santos, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, Caroline Horsch, and Praveen Ravi. Pettingzoo: Gym for multi-agent reinforcement learning. *CoRR*, abs/2009.14471, 2020. URL https://arxiv.org/abs/2009.14471.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023. URL https://zenodo.org/record/8127025.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *Journal of Machine Learning Research*, 23(267):1–6, 2022a. URL http://jmlr.org/papers/v23/21-1127.html.

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. Envpool: A highly parallel reinforcement learning environment execution engine, 2022b.