Towards verifiable Benchmarks for Reinforcement Learning

Matthias Müller-Brockhausen, Aske Plaat, Mike Preuss
Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
The Netherlands
{m.f.t.muller-brockhausen, a.plaat, m.preuss}@liacs.leidenuniv.nl

***Abstract*—Reinforcement Learning (RL) is one of the most dynamic research areas in Game AI and AI as a whole, and a wide variety of games are used as its prominent test problems. However, it is subject to the replicability crisis that currently affects most algorithmic AI research. Benchmarking in Reinforcement Learning could be improved through verifiable results. There are numerous benchmark environments whose scores are used to compare different algorithms, such as Atari. Nevertheless, reviewers must trust that figures represent truthful values, as it is difficult to reproduce an exact training curve. We propose improving this situation by providing access to the original evaluation data to validate study results. To that end, we rely on the concept of *replay* traces. These allow re-simulation of action sequences in deterministic RL environments and, in turn, enable reviewers to verify, re-use, and manually inspect evaluation results without needing large compute clusters. It also permits validation of presented reward graphs, an inspection of individual episodes, and re-use of result data (baselines) for proper comparison in follow-up papers. We offer plug-and-play code that works with Gym so that our measures fit well in the existing RL and reproducibility eco-system. Our approach is freely available, easy to use, and adds minimal overhead, as *replay* traces allow a data compression ratio of up to $\approx 10^4 : 1$ (94 GB to 8 MB for Atari Pong) compared to a regular MDP trace used in offline RL datasets. The paper presents proof-of-concept results for a variety of games.**

***Index Terms*—Verifiable, Benchmarks, Reinforcement Learning, Reproducibility**

I. Introduction

Reproducibility is a key component of peer-reviewed science. Reviewers are supposed to read, understand, and ideally be able to reproduce an experiment to ensure its factual correctness. It touches not only computer science, but any science, as without easy reproducibility, fraud is difficult to detect [1]. Especially for benchmarks and competitions, where fraudulent submissions potentially poison the rankings of a leaderboard, it is important to have tools for validation.

Benchmarking AI algorithms has become increasingly important and is now a driving force behind algorithm development. In Game AI, competitions have been an important part of scientific conferences for a long time already, and especially game problem benchmarks are currently more and more spreading out to core AI conferences, e.g., with the MineRL competition at NeurIPS [2], [3]. Whereas the overall aims of algorithm development are often to improve generality and especially sample efficiency, the employed methods are still relatively slow and thus need very long runs, thereby making reproducibility difficult.

In theory, computers are excellent for reproducibility. One can run the same code, bit for bit, on many different machines. This may be simplified down to issuing a single command based on technologies such as Docker [4]. However, methods that include some sort of non-determinism (e.g., training evolutionary algorithms [5], or deep neural networks [6]) hamper the reproducibility of experiments. Another growing problem is the availability of computing resources that would be needed to replicate results [7]. The tremendous successes of AlphaStar [8] and Dota 2 [9] are prominent examples. The large computing clusters they relied on are unavailable to most researchers for running any type of replication experiment. Furthermore, it becomes increasingly important to also consider sustainability issues, as the big cluster experiments are energy inefficient. Such considerations have been voiced, e.g., for Natural Language Processing (NLP) [10] or complex Games as Go (AlphaZero) [11]. It would thus make more sense to avoid re-computing everything but to improve the inspection of existing log data. Other issues that stand in the way of exact replication include insufficient reporting [12] or not open-sourcing code [13].

If replication itself is unavailable for some experiments, the next best thing could be verifiability, namely the ability to inspect, check, and replay parts of the evaluation. However, this is also difficult even in terms of handling the huge amounts of data that are produced during the big experiments. In order to achieve this, we would need some way of highly compressing this data, which instantly points us to the concept of *replay traces*.

The research question we are going to tackle in this work is thus: *How may a researcher verify the evaluation of a reinforcement learning algorithm of other researchers, especially the display of results in figures and tables, based on replay traces?*

We offer the following contributions:

- We explain how replay traces (Section II) allow reproducible verification of results such as benchmark leaderboards (Section III-A). Moreover, we empirically show that they enable a compression ratio of up to $10^4 : 1$ for offline RL datasets (Section IV-A)

- We provide Plug-n-Play code [14] to collect replay traces that integrate with the RL-Ecosystem (Gym [15])
- We provide an agenda for further research on how to obtain verifiable RL evaluation results using replay traces (Section V)

Although there are many factors at play with reproducibility, our work solely focuses on methodological improvements for reinforcement learning research. After briefly introducing the concept of replay traces (section II), we first look at suggestions that have already been made for improving reproducibility or experimental methodology in section III. Based on that state, we find improvable points (section III-A) and suggest concrete, actionable steps (section V). We then outline how these steps can be applied in practice with the code that we provide (section V-A). To enable compatibility, we ensured that it properly interfaces with the existing RL-Ecosystem.

II. Background: Replay Traces

For the following sections, the concept of replay traces is important. Thus we review its origins and known uses here.

Reinforcement learning optimizes sequential decision making processes, that are modeled as so called Markov decision processes (MDPs). An MDP consists of a tuple (S, A, $P_a(s, s')$, $R_a(s, s')$) (state space, action space, the probability of going from state s to s', and reward for going from s to s') [16].

A *trace*, also commonly referred to as an Episode within an RL-Environment [17], is a list of tuples that contain the start state s_t, the chosen action a, the received reward r, and the resulting state s_{t+1}. Traces are sufficient to train an RL Algorithm offline / off-policy, and they are also shared by related work as dataset-basis for training [18].

We introduce the term *replay traces*. Staying true to the computer science reinforcement learning terminology drawing inspiration from psychology [19], we were inspired by the work "Predicting the Past from Minimal Traces" [20]. We want reviewers to reliably predict (verify) the past (evaluation results) using traces that use up a minimal amount of storage space. To reduce the used space of traces, we assume that an MDP, given the same initial state s_0 and action sequence α, will yield the exact same trace. To reliably re-reproduce the initial state s_0 and follow up states, the random outcomes in $P_a(s, s')$, as well as parameters influencing the behavior of the environment need to be fixed based on an initial configuration s_{init}. Hence s_{init} contains the random seed as well as environment hyper parameters (see Table II for an example of CartPole hyper parameters). Fixing these probability outcomes is commonly referred to as a deterministic MDP [21]. Deterministic MDPs reduce the required data for re-simulation of replay traces to s_{init} and α. Replay traces fit well into reinforcement learning problems as there the action set A is usually smaller than the state space S. Hence it makes more sense to only save actions if the observations can be re-constructed afterward. While the added re-simulation cost might seem impractical for verification purposes, our experiments show that it can take less than 0.7% of the original RL training time (Section IV-A).

III. Related Work

While the matters of reproducibility, replicability, and verifiability are relevant to all scientific fields [1], we will focus on reinforcement learning here. In reinforcement learning, previous works suggest guidelines on how to design and report a well reproducible experiment [12]. Conferences such as NeurIPS are moving towards implementing these guidelines and ask reviewers to fill in a questionnaire about reproducibility. This has led to more and more sharing of code, and researchers are encouraged to do so [13]. Moreover, reviewers found it easier to judge submissions that included code. Most of the reviewer guidelines focus on the paper itself, which is the well-established scientific tradition that was practiced already when computers were not yet invented. However, many researchers in Computer Science now believe that for experimental works, we should go one step further and exploit its theoretically perfect and exact ability to verify the factual correctness of reported results and submitted code / data (Section III-A). This so-called "Verification of Artifacts" [5] is not a new concept. For example, tools available to make policy training as reproducible as possible are readily available (e.g., Garage [22]). For experiments that are not using tools such as Garage, there are also clear guidelines on how to properly compare to a baseline of an algorithm [23]. Moreover, researchers have suggested saving the final values used in graphs to be able to verify the figures [24]. This theoretically works for any Figure. However, how can we be sure that the Figure is correct [25]?

Games are an interesting playground for RL-AI. GVGAI is a prominent example [26], as it is used for competitions that benchmark individual algorithm submissions from both planning [27] and learning,m such as RL [28]. For these competitions, the validity of results is guaranteed by means of execution of the submitted code by the event host. This is made possible through a pre-defined agent interface that allows interacting with arbitrary games. In other reinforcement learning environments, we also have a pre-defined agent interface (see the center of Figure 1), but re-running policy training is not at all reproducible. Reproducibility is not guaranteed in GVGAI either, as the applied algorithms, such as MCTS [29], include randomness that is not fixed. While GVGAI remedies this by means of multiple runs, research has shown that averaging over runs does not prevent inconsistencies in reproduction attempts [12]. Whereas replay traces do not alleviate the problem directly, they do lift the requirement to have to run the agent code oneself.

A. Why replay traces?

Reproducibility in the RL ecosystem is an evolving matter. Environments usually behave deterministically if seeded as, e.g., CartPole or MountainCar in Gym [15]. Famous problems that did not yet satisfy this requirement have been converted (e.g., Robotics [30], [31]). Moreover, besides theory-focused

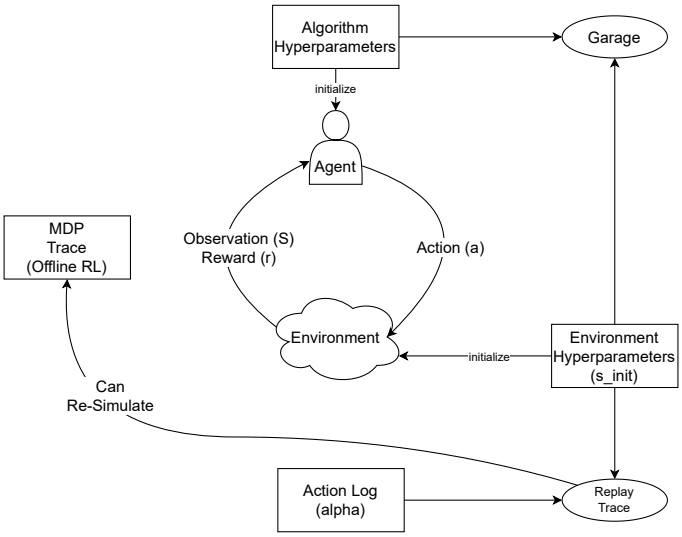

Fig. 1: A visualization of the relationship between reinforcement learning training, offline RL datasets (traces), replay traces Garage [22], as well as the data collected for each.

guidelines [12], practical tools like Garage exist to enable reproducible policy training [22].

Nonetheless, RL lacks verifiability of evaluation results, such as benchmark submissions. Reviewers shall be enabled to easily verify reported results in figures or tables with minimal effort. MDP's already come with the concept of *traces* that are basically a full log of all data (observation, action, reward) [17], from which figure data could also be constructed (see Section II). These traces are used for offline reinforcement learning. However, offline RL datasets can become quite large (3TB for the experiments in one paper [18]) and, as a consequence, are hosted on a proprietary central authority. To remedy this problem, we collect as little data as possible and without requiring a central hosting authority regardless of size (Section IV-C). In Figure 1, we attempt to visualize the relationship between offline RL datasets, replay traces, and the data that Garage [22] collects. Thus, we suggest collecting an initial environment configuration corresponding to s_{init} from the Replay Trace (Section II). In Gym, s_{init} contains all values that influence an environments initialization (reset) and transition function (step). Table II contains an example of s_{init} for CartPole. Moreover, the actions that an agent takes are saved. A consequence is that our method is limited to fully deterministic environments. By properly seeding non-deterministic algorithms' random number generation, non-deterministic problems can also be used.

We later show that replay traces allow a compression ratio of up to 77 for regular and up to 12559 for image-based environments (Section IV-A).

Our focus on the environment instead of the policy training is well motivated. The data collected by tools such as Garage [22] can not overcome one specific problem. The training of neural networks includes certain operations that render it not reproducible if the host machine, host operating system, or software library version change [32]. Unless researchers have the exact same machine and software configuration, reproducibility will become difficult. Software configurations are easily reproducible via Docker [4]. However, if the same code produces different results on other machines, then reproducibility becomes practically impossible.

We see replay traces as an add-on rather than a replacement to current reproducibility approaches. It is a workaround because reproducible policy training, such as Garage [22] attempts to offer, does not yet work. Should training become fully deterministic, replay traces might become obsolete. Nevertheless, they would still offer the advantage of being computationally cheaper than training, as environment execution requires less computational resources than updating weights of a large neural network. Moreover, environment re-simulation is cheap enough to potentially run in the web browser enabling interactive tools, which is difficult to reproduce with computing clusters needed for training.

Whereas replay traces are limited to reinforcement learning, their concept transfers perfectly to video games. TrackMania is a great example because its physics is fully deterministic. For its leaderboard, replays are saved. A replay contains the name of a level and all taken actions by the player. Leaderboard submissions are verified for validity [33]. Moreover, it allows to detect Tool-Assisted Runs in pure human play leaderboards [34]. More complex games, such as Counter Strike: Global Offensive, Dota 2, and Starcraft, support replays as well. Yet, their replay formats contain a multitude of possible inputs that are difficult to map directly to a reinforcement learning action space and hence replay traces. The same applies to the Unreal Engine, which has a general ReplaySystem that can replay any data [35] but does not automatically allow replaying arbitrary scenes deterministically based purely on agent actions. Nevertheless, this larger trove of data is still useful for detecting cheats, such as AimBots [36]. Furthermore, it can be used to make estimations of player skill [37].

As games are used in competitions, they could also be applied for validation there. For example, two of the games we previously mentioned to support replays have been a competition at the IEEE Conference on Games 2021: Dota 2 [38] and Starcraft [39]. Moreover, SpaceInvaders, which we test from Gym, is also a CoG competition [40]. Other games seem suitable as well, such as GvGAI [27], Snakes [41], or Bot Bowl [42]. These games might even manage to be directly compatible with replay traces, as, for example, GvGAI has managed to provide an RL-Gym Environment for its learning track [28].

Lastly, our data collection suggestion harmonizes with the concept of Procedural Content Generation (PCG), as the seed can be saved for deterministic reproduction. ProcGen has already shown that current RL-Algorithms are struggling to generalize [43]. However, PCG applied correctly already enables better generalization [44] and more fine-grained training curricula [45], and is thus especially important in benchmarking Transfer in Reinforcement Learning [46].

IV. Method

We will describe our approach in detail, along the lines of 3 different aspects. Replay traces achieve a high compression ratio compared to regular traces. They can compress up to $10^4 : 1$ (Section IV-A). Next, we detail how replay traces enable re-usable visualizations (Section IV-B). Moreover, we suggest the usage of a distributed file system, the interplanetary file system (IPFS) [47], for long-term storage (Section IV-C). Based on these insights, we propose a reproducibility agenda (Section V).

A. Data compression

Replay traces are a way to store evaluation result signatures efficiently. We will have a closer look at efficient storage for different games. Please refer to Table I for the size comparisons.

Reinforcement learning traces can take up a considerable amount of space. For example, a single work offering an offline RL trace provides 3TB of data [18]. In order to compare the amount of space needed for traces vs. replay traces (Section II), we chose different environments with growing observation spaces. Taxi with one integer, CartPole with two floats, BipedalWalker with 24 floats, Atari-Games RAM using 128 bytes, and the produced 210 x 160 pixel RGB image using 100800 bytes. More interestingly, whereas most environments have only a single integer action space A, BipedalWalker has three continuous actions increasing the space required for replay traces. We train on the environment for 1 Million steps using PPO [48] from stable baselines 3 [49], using the default hyperparameters for both environment and agent. During training, we collect the full MDP-trace (Observation, Action, and Reward) and the replay MDP-trace (Env-Params and Actions) for each environment. The results in Table I are striking: Replay traces enable a compression ratio of 12559.36 for image-based environments with a single action, such as Atari Pong. For the 128-byte ram observation, the compression ratio falls to 99.58 for Pong, reducing 767.18 MB to 7.7 MB.

Nevertheless, environments with small observation spaces such as CartPole still allow a compression rate of 53, reducing a 452.25 MB trace down to 8.5 MB. However, BipedalWalker underlines that the potentially saved space depends solely on the size difference between the observation space S and the action space A. For BipedalWalker, 24 numbers in the observation space vs. 4 in the action space. Hence the compression ratio of 2.9 reduces the 530.72 MB trace down to 181.79 MB. An intriguing discovery we made is a varying size for the re-simulated trace of BipedalWalker, which should not occur. Thus we performed an experimental analysis and found that 5 in 100 re-simulations yielded a different trace due to rounding errors. Consequently, BipedalWalker is not yet fully deterministic. We repeated this experiment for all other tested environments in Table 1 and found that they are fully deterministic, hence yielding 0 failed re-simulations.

We also measured the time to re-simulate a full MDP-trace from a replay MDP-trace vs. the training time. Our measurements are also shown in Table II. Note that for timing-related data, there is variation in runs for different hardware. All experiments were run on a machine with an Intel Xeon Silver 4214, an Nvidia Geforce RTX 3090, and 256GB of RAM. The cost of re-simulation depends on the complexity and observation space of the environment. We make use of multi-threading to re-simulating multiple episodes at once. For Atari, re-simulation varies per observation and game. In the worst case, it takes 22.39% of the training time for Breakout-ram-v0, and in the best case 6.03% for Pong-v0. Computationally less intensive environments, such as BipedalWalker-v3 or CartPole can lower this further to less than 1 % (0.68%) of training time.

B. Re-Usable Visualizations

```
{"$schema": "...",
        "description": "Reward per Episode",
        "data": {"values": [{"Episode": 0,
                             "Reward": 11.0, "env": "..."
                                }, ...]},
        "mark": "line",
        "encoding": {
                "x": {"field": "Episode",
                             "type": "quantitative"},
                "y": {"field": "Reward",
                             "type": "quantitative"},
                "color": {"field": "env",
                             "type": "nominal"}}
}
```

Listing 1: An excerpt for a re-usable Vega Reward Graph description. Generates the graph shown in Figure 2. The full JSON-File that can be explored in the Vega-Editor can be found in the Code-Repository as listing_graph.json.

Khetarpal et al. [24] suggest saving the values that are used to plot figures and providing the code to the plot. This improves reproducibility when the numbers are the results of the experiments. Replay traces enable re-simulating this data to verify if this is the case. Moreover, replay traces allow looking at different values than those presented in a paper. For example, if a paper reported only the average reward, one could extract the median reward instead through the re-simulated data. Alternatively, if reward per episode was reported, one could instead look at reward per step. To increase the re-usability and accessibility of figures, we suggest using Vega [50]. Vega is a JSON-based graph description language. The main advantage is that plotted values are embedded inside the human-readable JSON data. Hence, one could extract a baseline algorithm reward line from the Vega description of an original paper and then compare it to a newly trained variation without re-simulation. Of course, it still allows exporting a scalable graphic for usage inside of the paper (e.g., Figure 2). In this case, guidelines on designing a proper baseline [23] are not that important anymore because the actual data from other papers can be directly compared[1].

Another advantage of Vega [50] is the ability to load Graph-Definition-Files in a Browser. These come with various

[1]Assuming they are being compared on the same benchmark environment with the same configuration

Environment	Trace (MB)	Replay Trace (MB)	Compression Ratio	Training (sec)	Re-simulation (sec)	% Re-Simulation to Training
Assault-ram-v0	767.21	7.76	98.82	2314.0	177.0	7.65
Assault-v0	96165.33	7.78	12355.67	5372.0	393.0	7.32
BipedalWalker-v3	530.72	181.79	**2.90**	2241.0	15.0	0.67
Breakout-ram-v0	757.55	8.33	90.94	2859.0	640.0	**22.39**
Breakout-v0	96504.98	7.98	12100.5	5647.0	520.0	9.21
CartPole-v0	452.25	8.5	53.23	1907.0	13.0	**0.68**
Pong-ram-v0	767.18	7.7	**99.58**	2356.0	142.0	**6.03**
Pong-v0	94641.99	7.54	**12559.36**	5319.0	321.0	6.03
SpaceInvaders-ram-v0	767.45	7.76	98.91	2412.0	181.0	7.5
SpaceInvaders-v0	95973.03	7.75	12378.43	5448.0	378.0	6.94
Taxi-v3	335.85	8.46	39.69	2053.0	68.0	3.31

TABLE I: A comparison of the space requirements in megabytes and re-simulation cost in seconds for (replay) traces when training PPO for 1 Million steps and averaged over three runs. We see that replay traces achieve high compression ratios for single action environments such as Atari or CartPole. Even in the worst case (BipedalWalker-v3), replay traces still allow a compression ratio of 2.9, saving nearly 350 MB.

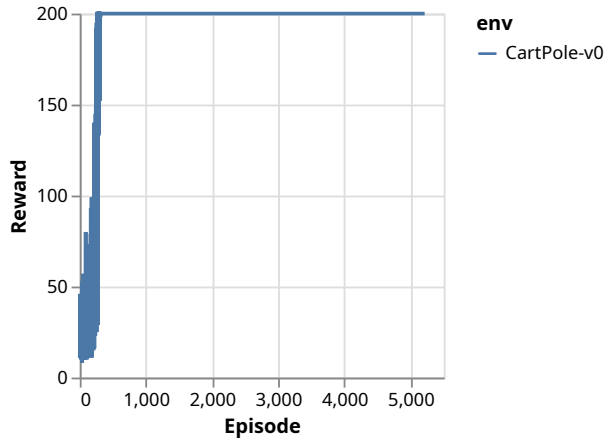

Fig. 2: A re-usable Vega Lite Reward Graph, generated from the Description shown in Listing 1. It displays the sum of reward per episode achieved during training of PPO3 [48] for 1 million steps using the default hyperparameters for algorithm and environment. The used reward data can be re-simulated based on the replay trace with ID "2IEpetGGwN-FOs38rhAFHx".

advantages, such as seeing the exact value of an individual coordinate, zooming, scrolling through the axes, and changing colors if these are not color-blind friendly.

C. Data availability

The data we suggest to collect (Section III-A) potentially requires much storage. Whereas some hosting authorities allow researchers to upload large datasets for long-term availability, the two main problems with a central authority are the possibility that data could be tampered with / changed and that there is only a single point of failure regarding availability. The Interplanetary File System [47] lets us address these issues. It behaves similar to BitTorrent. Users who downloaded a file also share it, eliminating the need for a central server to store all files. Moreover, the data is hashed, so it can not be altered afterward by the original author, the hosting authority, or a redistributing user. A hash can represent a full folder with many subfiles that also all have individual hashes / IDs. So the log file itself and the results produced with it become verifiable. IPFS [47] also advertises its usefulness for scientific purposes, and we believe our replay traces are well suited for it. Moreover, IPFS can co-exist and even integrate into a hosting service (such as Zenodo [51]) that could mirror the data it provides via IPFS. That would allow them to use it as a Content Delivery Network (CDN) for the actual files listed in the Zenodo database. Finally, conferences or journals could have public lists of relevant dataset IDs for published papers that should be *pinned*. Pinning in IPFS can be seen as the equivalent of mirroring data. A pinned ID will permanently (until unpinned) be held in local storage and thus be available to others requesting it.

V. Reproducibility Agenda

Replay traces are a useful step on the path to reproducibility, allowing efficient verification of evaluation data. To further improve the reproducibility of the field and to put our work in perspective, we suggest the following agenda.

We propose researchers experimenting with deterministic Environments (ideally using PCG in order to improve generalizability) to do the following:

1) Use the verifiability tool to collect replay traces
2) Provide source code, including a runnable container (e.g., Docker), to allow verification of results and figures
3) Generate figures using a common visualization grammar such as Vega [50], facilitating re-use of figure data
4) Utilize IPFS [47] to ensure the data's integrity and long-term availability

A. Agenda applied to CartPole

To better illustrate our suggested agenda in practice, we will illustrate how following it looks for the proverbial CartPole environment. In Table II, we have prepared the environment hyperparameters (s_{init}) relevant for each CartPole episode. Garage [22] (Figure 1) also collects these parameters but assumes policy training to be fully reproducible, which it is currently not (Section III-A). In Listing 2, we show that recording replay traces merely requires wrapping the Gym-environment. The code in this listing is not pseudo-code but

Name	Default	Description
Gravity	9.8	Gravitational Power of the Planet
Pole Mass	0.1	The mass of the pole balancing on the cart
Pole Length	0.5	The length of the pole (The actual length is this value times two)
Cart Mass	1	The mass of the cart
Force Magnitude	10	Strength of the force applied to the cart on input
Tau	0.02	How much time passes between state updates (in seconds)
Theta Threshold Radians	∼0.209 (∼12°)	The angle at which the pole is considered to be tipped over
X Threshold	2.4	The area in which the cart may move without being considered out of bounds
Use euler kinematics	TRUE	Influences cart movement
Random Seed	varies	Influences initial cart position and pole angle

TABLE II: Complete list of initial Environment Parameters s_{init} (Section II) for Cartpole.
Given the same s_{init} and action sequence α, a deterministic MDP like CartPole will always yield the same trace.

the actual main file of our runnable example. The last function generates a Vega [50] (Section IV-B) JSON-Description (Listing 1) that can be used to visualize a graph (Figure 2). It can also be pasted into the Vega-Editor[2]. Table I is also based on re-simulated data from replay traces.

New projects wanting to record replay traces need to wrap their Gym-environment with the *record* function from the *vgym*-folder [14] before training. Then, the replay traces will be saved into a sub-folder of the current working directory. A replay trace file is serialized into the Concise Binary Object Format (CBOR) [52] and compressed with zlib [53]. Note that replay trace sizes in Table I reflects the size before serialization and compression. The utility function *load_replay* in our package abstracts these serialization details away from the user. After loading a file, either all episodes can be re-simulated into regular traces in parallel using *resimulate_parallel*, or an individual episode can be re-simulated using *episode_to_trace*. Based on that, re-simulated data figures and tables should be generated, and the code as well as recorded replay traces published alongside the submission.

```
import gym
from vgym import record, resimulate
from stable_baselines3 import PPO
from example import make_graphs_from
env = gym.make("CartPole-v0")
renv = record(env, name="CartPole-v0")
renv.reset()
PPO("MlpPolicy", renv, verbose=1)
.learn(1000000)
# Recreate Trace from replay trace
trace = resimulate(renv,
renv.minimal_traces.to_list())
# Plot using re-simulated data
make_graphs_from([("CartPole-v0", trace)])
```

Listing 2: Example code that trains a policy on Cartpole collecting a replay trace. The full trace is re-simulated based on the replay trace and used to generate a Vega Graph-Description.

Our code repository contains a Dockerfile, recorded / used replay traces, the example code behind the imported functions of Listing 2, and a readme detailing how to re-simulate the shown graph (Figure 2), as well as the values for the size comparison table (Table I). All data is hosted on IPFS [47] here [14].

[2]https://vega.github.io/editor/

VI. Discussion

If properly applied, the concept of replay traces allows for reproducible and verifiable reinforcement learning evaluations. Moreover, it enables re-usability of result data, more accessible ways to view the data interactively, inspection of individual episodes, and data-saving for offline RL datasets.

A limitation of replay traces is that the source of the action sequence can not properly be verified. From the obtained data alone, one cannot exclude any sophisticated ways of cheating the authors may have applied. Whereas a result figure or table may be fully reproduced with our data, one can not know whether the agent created the supplied action sequences. The data could be handcrafted or made by a heuristic or any other method that is not listed in the reviewed paper. Although replay traces do not yet achieve end-to-end verifiable research experiments, they are an important step towards verifiable evaluation figures to show that values portrayed in figures or tables have truly been achieved within the tested environment and have not been randomly generated. In combination with host executed competition benchmarks such as GVGAI, they enable post hoc analysis of individual agent performances.

A. Environmental Impact

In times where big research experiments use large amounts of electricity [54], it is important to note the possible environmental impact of our suggestion. Hard-disk space seems cheaper than the high wattages of training an agent on a GPU. Hard-disk space instead of computation is also used to "greenwash" novel blockchains as Chia [55]. Nevertheless, nothing electronic comes free, so the hardware still needs to be built, and energy needs to be used to power the servers pertaining the datasets we suggest to collect. In consequence, it will increase the global environmental impact of RL. However, we see no other less intrusive way to ensure verifiability of RL evaluation results. Thanks to the suggestion to share the data via IPFS [47], the servers would not need to stay online 24 / 7. There could be specific times of data availability where the server is switched on and the data accessible while ensuring the data is not tampered with. Moreover, the distributed nature of IPFS might make the necessity of a central storage server

obsolete, given enough participants pertaining parts of the datasets.

VII. Conclusion

In reinforcement learning, reproducibility of experimental results, and verification of research claims, is an important challenge. Our work introduces a methodology to verify evaluation results building on the concept of replay traces. We provide a full implementation of this method and have tested it on small and larger reinforcement learning experiments.

For typical experiments, replay traces enable compression ratios of up to 12539, reducing an offline RL trace of Atari Pong from 94 GB down to 8 MB. Moreover, re-simulating this replay trace back to its original size takes 6% of the original training time.

As our example shows, the collection of replay traces requires but a wrapper around a Gym-environment.

While replay traces are limited to deterministic Reinforcement Learning problems, the idea transfers well to (video) games. Trackmania already applies leaderboard verification via replays, showing that benchmarks and competitions could adopt similar concepts. We envision a web-based tool that allows re-simulation and verification of results without any software setup by a reviewer on their local machine for future work. To that end, we provide a mock-up (Figure 3) with a functionality description. This could be implemented through either a Rust-Gym port of the current approach or by having a Python interpreter that properly works in a web assembly environment and with Gym.

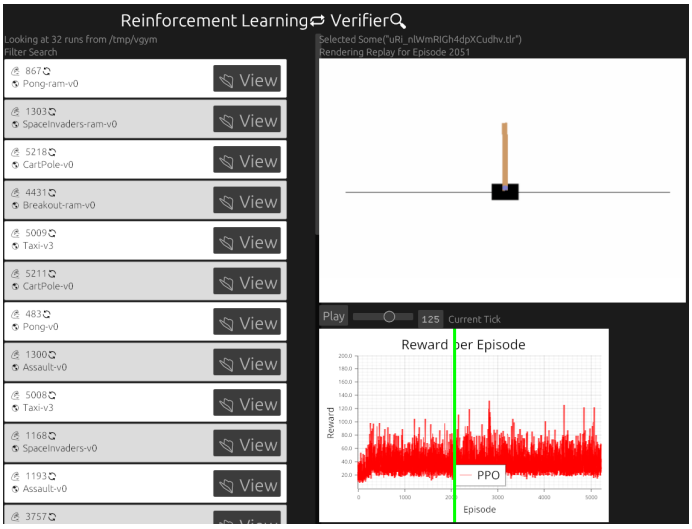

Fig. 3: Mock-up of a web-based tool we envision for future work. The left side contains a list of replay traces, the environment, and amount of contained episodes. The right side shows details of the re-simulated trace. For inspection, one can select individual episode replays directly from the training reward graph, highlighted by the green vertical stripe. Each episode can be stepped through frame by frame.

References

[1] National Academies of Sciences, Engineering, and Medicine and others, *Reproducibility and replicability in science*. National Academies Press, 2019.

[2] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. S. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, M. Veloso, and P. Wang, "The minerl competition on sample efficient reinforcement learning using human priors," in *Thirty-third Conference on Neural Information Processing Systems (NeurIPS) Competition track*, December 2019. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-minerl-competition-on-sample-efficient-reinforcement-learning-using-human-priors/

[3] W. H. Guss, M. Y. Castro, S. Devlin, B. Houghton, N. S. Kuno, C. Loomis, S. Milani, S. Mohanty, K. Nakata, R. Salakhutdinov, J. Schulman, S. Shiroshita, N. Topin, A. Ummadisingu, and O. Vinyals, "The minerl 2020 competition on sample efficient reinforcement learning using human priors," January 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/the-minerl-2020-competition-on-sample-efficient-reinforcement-learning-using-human-priors/

[4] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[5] M. López-Ibáñez, J. Branke, and L. Paquete, "Reproducibility in evolutionary computation," *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 4, pp. 1–21, 2021.

[6] C. Liu, C. Gao, X. Xia, D. Lo, J. Grundy, and X. Yang, "On the replicability and reproducibility of deep learning in software engineering," *arXiv preprint arXiv:2006.14244*, 2020.

[7] J. S. O. Ceron and P. S. Castro, "Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1373–1383.

[8] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft II using multi-agent reinforcement learning," *Nat.*, vol. 575, no. 7782, pp. 350–354, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1724-z

[9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019. [Online]. Available: http://arxiv.org/abs/1912.06680

[10] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," 2019.

[11] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[12] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[13] J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle, "Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program)," *Journal of Machine Learning Research*, vol. 22, 2021.

[14] A. paper authors. (2022-02-27) Code and trace repository. [Online]. Available: https://ipfs.io/ipfs/QmRDg98PaQEdvrj6tcDr5eQRLCPc2YovphMd7uHNJet1Ug

[15] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[16] R. BELLMAN, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. [Online]. Available: http://www.jstor.org/stable/24900506

[17] A. Plaat, *Deep Reinforcement Learning*. Springer Nature, 2021.

[18] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *International Conference on Machine Learning*, 2020.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction. Second Edition*. MIT press, 2018.

[20] M. Werning, "Predicting the past from minimal traces: Episodic memory and its distinction from imagination and preservation," *Review of philosophy and psychology*, vol. 11, no. 2, pp. 301–333, 2020.

[21] I. Post and Y. Ye, "The simplex method is strongly polynomial for deterministic markov decision processes," *Mathematics of Operations Research*, vol. 40, no. 4, pp. 859–868, 2015.

[22] T. garage contributors, "Garage: A toolkit for reproducible reinforcement learning research," https://github.com/rlworkgroup/garage, 2019.

[23] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," 2017.

[24] K. Khetarpal, Z. Ahmed, A. Cianflone, R. Islam, and J. Pineau, "Re-evaluate: Reproducibility in evaluating reinforcement learning algorithms," 2018.

[25] D. Eisner, "Reproducibility of science: Fraud, impact factors and carelessness," *Journal of molecular and cellular cardiology*, vol. 114, pp. 364–368, 2018.

[26] D. Pérez-Liébana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the robustness of general video game playing agents," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2016, pp. 1–8.

[27] R. D. Gaina, A. Couëtoux, D. J. Soemers, M. H. Winands, T. Vodopivec, F. Kirchgeßner, J. Liu, S. M. Lucas, and D. Perez-Liebana, "The 2016 two-player gvgai competition," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 209–220, 2017.

[28] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game ai," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2018, pp. 1–8.

[29] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[30] D. Ferigo, S. Traversaro, G. Metta, and D. Pucci, "Gym-ignition: Reproducible robotic simulations for reinforcement learning," in *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2020, pp. 885–890.

[31] P. Aumjaud, D. McAuliffe, F. J. R. Lera, and P. Cardiff, "rl_reach: Reproducible reinforcement learning experiments for robotic reaching tasks," *Software Impacts*, vol. 8, p. 100061, 2021.

[32] PyTorch. (2021-12-15) Reproducibility - pytorch documentation. [Online]. Available: https://pytorch.org/docs/stable/notes/randomness.html

[33] A. Donadigo. (2021-12-15) Extracting inputs from replays. [Online]. Available: https://donadigo.com/tminterface/input-extraction

[34] ——. (2021-12-15) Tmx replay investigation. [Online]. Available: https://donadigo.com/tmx1

[35] E. Games. (2021-12-15) Replay system - unreal engine documentation. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Replay/

[36] K. Maberry, S. Paustian, and S. Bakir, "Using an artificial neural network to detect aim assistance in counter-strike: Global offensive," *DOI*, vol. 10, no. 1235, pp. 1–4.

[37] P. Xenopoulos, H. Doraiswamy, and C. Silva, "Valuing player actions in counter-strike: Global offensive," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 1283–1292.

[38] J. M. Font and T. Mahlmann, "Dota 2 bot competition," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 285–289, 2018.

[39] S. S. Farooq, I.-S. Oh, M.-J. Kim, and K. J. Kim, "Starcraft ai competition report," *AI Magazine*, vol. 37, no. 2, pp. 102–107, 2016.

[40] J. A. Brown, L. J. P. de Araujo, and A. Grichshenko, "Ai space invaders 2021 competition," 2021.

[41] ——, "Snakes ai competition 2020 and 2021 report," 2021.

[42] N. Justesen, P. D. Moore, L. M. Uth, J. Togelius, C. Jakobsen, and S. Risi, "Blood bowl: A new board game challenge and competition for ai," in *2019 IEEE Conference on Games (COG)*. IEEE, 2019.

[43] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," *arXiv preprint arXiv:1912.01588*, 2019.

[44] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 428–436, 2020.

[45] M. C. Green, B. Sergent, P. Shandilya, and V. Kumar, "Evolutionarily-curated curriculum learning for deep reinforcement learning agents," 2019.

[46] M. Müller-Brockhausen, M. Preuss, and A. Plaat, "Procedural content generation: Better benchmarks for transfer reinforcement learning," in *2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021*. IEEE, 2021, pp. 1–8. [Online]. Available: https://doi.org/10.1109/CoG52621.2021.9619000

[47] S. Muralidharan and H. Ko, "An interplanetary file system (ipfs) based iot framework," in *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 2019, pp. 1–2.

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

[49] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[50] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 341–350, 2016.

[51] European Organization For Nuclear Research and OpenAIRE, "Zenodo," 2013. [Online]. Available: https://www.zenodo.org/

[52] C. Bormann and P. Hoffman, "Concise binary object representation (cbor)," RFC 7049, DOI 10.17487/RFC7049, October 2013,¡ https://www. rfc-editor. org . . . , Tech. Rep., 2013.

[53] P. Deutsch and J.-L. Gailly, "Zlib compressed data format specification version 3.3," RFC 1950, May, Tech. Rep., 1996.

[54] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," 2021.

[55] B. Cohen and K. Pietrzak, "The chia network blockchain," 2019.