2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)

On the Mistaken Assumption of Interchangeable Deep Reinforcement Learning Implementations

Rajdeep Singh Hundal
National University of Singapore
Singapore
rajdeep@u.nus.edu

Yan Xiao*†
Sun Yat-sen University
Shenzhen, China
xiaoy367@mail.sysu.edu.cn

Xiaochun Cao†
Sun Yat-sen University
Shenzhen, China
caoxiaochun@mail.sysu.edu.cn

Jin Song Dong
National University of Singapore
Singapore
dcsdjs@nus.edu.sg

Manuel Rigger
National University of Singapore
Singapore
rigger@nus.edu.sg

***Abstract*—*Deep Reinforcement Learning* (DRL) is a paradigm of artificial intelligence where an *agent* uses a neural network to learn which actions to take in a given *environment*. DRL has recently gained traction from being able to solve complex environments like driving simulators, 3D robotic control, and multiplayer-online-battle-arena video games. Numerous *implementations* of the state-of-the-art algorithms responsible for training these agents, like the *Deep Q-Network* (DQN) and *Proximal Policy Optimization* (PPO) algorithms, currently exist. However, studies make the mistake of assuming implementations of the same algorithm to be consistent and thus, *interchangeable*. In this paper, through a *differential testing* lens, we present the results of studying the extent of implementation inconsistencies, their effect on the implementations' performance, as well as their impact on the conclusions of prior studies under the assumption of interchangeable implementations. The outcomes of our differential tests showed significant discrepancies between the tested algorithm implementations, indicating that they are *not* interchangeable. In particular, out of the five PPO implementations tested on 56 games, three implementations achieved superhuman performance for 50% of their total trials while the other two implementations only achieved superhuman performance for less than 15% of their total trials. Furthermore, the performance among the high-performing PPO implementations was found to differ significantly in nine games. As part of a meticulous manual analysis of the implementations' source code, we analyzed implementation discrepancies and determined that code-level inconsistencies primarily caused these discrepancies. Lastly, we replicated a study and showed that this assumption of implementation interchangeability was sufficient to *flip* experiment outcomes. Therefore, this calls for a shift in how implementations are being used. In addition, we recommend for (1) replicability studies for studies mistakenly assuming implementation interchangeability, (2) DRL researchers and practitioners to adopt the differential testing methodology proposed in this paper to combat implementation inconsistencies, and (3) the use of large environment suites.**

***Index Terms*—reinforcement learning, differential testing**

I. Introduction

Deep Learning (DL) and *Deep Reinforcement Learning* (DRL) are popular paradigms of *Artificial Intelligence* (AI) that use neural networks to solve a problem. Different from DL, where the dataset is fixed and re-used throughout the training process, DRL allows for online learning in a controlled *environment* where the dataset is not fixed but rather procured from the environment on-the-fly—the very reason it is chosen over DL in some scenarios such as video games and autonomous driving [1]–[3]. More formally, DRL is a paradigm of AI where a program, or more specifically, an *agent*, learns the optimal action to take in an environment after iterating multiple times through it. DRL has proven to be extremely effective in games with recent advances in the field; for example, DeepMind's AlphaZero [4] became the first algorithm to beat a world-champion computer program at Chess, Shogi, and Go and OpenAI's OpenAI Five [5] became the first algorithm to defeat the human world champions at the popular online video game Dota 2. Furthermore, DRL's applications span beyond those of games, from time-dependent systems like autonomous vehicles [6] and stock trading [7] to precision-focused systems like 3D robotic control [8], [9].

Similar to DL *libraries* like TensorFlow and PyTorch, numerous DRL libraries providing algorithm *implementations* have been created. With more than 10K GitHub stars, RLlib [10], Baselines [11], and Dopamine [12] are among the most popular DRL libraries. RLlib from Ray specialises in the distributed training of DRL algorithms, at a scalable level. Baselines from OpenAI and Dopamine from Google are research-focused libraries, built towards the quick replication and refinement of DRL algorithms.

In DRL studies, it is common to conduct comparisons between different algorithms [8], [9], [13], [14]. Moreover, as multiple implementations of an algorithm currently exist, some comparative studies use unoriginal implementations [15]–[18]. This is because researchers—as well as practitioners—assume that an algorithm would perform equally well across its implementations and thus, use them *interchangeably*. To demonstrate this, we systematically conducted a literature review which included research papers from two popular AI conferences and identified 23 research papers making this

*Corresponding author.
†Shenzhen Campus of Sun Yat-sen University.

1558-1225/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSE55347.2025.00222

assumption, with publishing dates ranging from 2017 to 2024.

DRL algorithms are *not* guaranteed to be consistent. On the contrary, they have a *high* risk of inconsistencies. Firstly, this is due to the large amount of *hyperparameters*—tunable values that guide and control the learning process. Being a combination of both DL and *Reinforcement Learning* (RL), DRL inherits hyperparameters from both of them. Secondly, because of the large amount of hyperparameters, DRL libraries run a higher risk of (1) making mistakes, and (2) improving or worsening algorithms via seemingly minor code-level implementation choices [19]. This results in DRL libraries implementing their own *flavour* of an algorithm, leading to similar but distinct implementations of the same algorithm in the DRL domain.

We conducted a pilot study to attain an *initial* assessment on the extent and effects of the aforementioned implementation inconsistencies. In particular, we compared five implementations of the *Deep Q-Network* (DQN) algorithm [20] and observed performance discrepancies as well as code-level inconsistencies. If a mature algorithm like DQN, which has already been extensively studied, standardized, and built upon, exhibits such disparity across some of the most popular implementations, it raises concerns about (1) the level of inconsistency and potential issues with the implementations of less mature algorithms, and (2) the validity of existing research that assume interchangeable algorithm implementations. Varying efficacies of the *same* algorithm across its implementations might render comparisons from studies that assume interchangeability invalid. This could have wide-ranging effects, as the conclusions of many studies might need to be re-examined.

In this paper, we assess the extent of implementation inconsistencies in the context of DRL and how they affect research under the assumption that implementations are consistent and interchangeable. At a high-level, our study relies on *differential testing*, which is a well-known software testing methodology [21]. In particular, we conducted a large-scale study (of approximately 10K GPU hours) to compare the efficacy of multiple implementations of the same algorithm to identify potential inconsistencies. Using differential testing, we answer the following research questions:

- **RQ1: How prevalent are discrepancies between implementations of the same algorithm?** Varying efficacies of an algorithm across its implementations would produce untrustworthy results and conclusions from studies that assume implementation interchangeability—using alternate implementations over the original. Thus, it is important to study the extent of these variances first before assuming that implementations are interchangeable. We answered RQ1 by using differential testing and state-of-the-art DRL comparison techniques to assess different implementations of the *Proximal Policy Optimization* (PPO) algorithm [9] in terms of efficacy (*i.e.,* mean reward), with statistical guarantees.
- **RQ2: Why do implementation discrepancies occur?** Understanding why implementation discrepancies occur is vital to creating effective solutions. Thus, we answered RQ2 by investigating the root cause of the discrepancies found in RQ1. In particular, similar to the pilot study, we inspected the implementations' source code for inconsistencies that accounted for the discrepancies found.
- **RQ3: Can the assumption of interchangeable implementations alter the outcomes of an experiment?** There would be cause for concern if using a different algorithm implementation was significant enough to *flip* experiment outcomes, as studies which assumed implementation interchangeability might then need to be re-examined. Thus, to determine if this was possible, we answered RQ3 by replicating a study that assumed implementation interchangeability [16], but with a different *Deep Deterministic Policy Gradient* (DDPG) implementation [22], and compared the outcomes.

Our experiments showed that DRL implementations were *mistaken* to be interchangeable, leading to significant alterations in experimental outcomes. In particular, out of the five PPO implementations tested on 56 environments, three implementations consistently outperformed the other two implementations in all comparison techniques used. Moreover, we statistically show that even the high-performing PPO implementations differed significantly among themselves in nine environments. When investigating the root cause of these discrepancies, we found multiple code-level inconsistencies that accounted for the discrepancies found. Lastly, our experiments showed that this assumption of implementation interchangeability was also significant enough to alter experiment outcomes by *flipping* two out of the three outcomes tested from the replicated study.

Thus, we recommend for (1) replicability studies for studies mistakenly assuming implementation interchangeability, (2) DRL researchers and practitioners to adopt the differential testing methodology proposed in this paper to combat implementation inconsistencies, and (3) the use of large environment suites. Furthermore, a key implication for researchers studying software testing is that, despite DRL's stochasticity and non-deterministic output, differential testing still can be applied. Therefore, the techniques used in this paper are potentially applicable to non-AI stochastic systems that can benefit from differential testing. Our source code is publicly available and can be found at https://doi.org/10.5281/zenodo.14249024.

II. Background

In this section, we discuss the necessary terminology used in this study.

a) Environment and agent: RL has two basic components, the agent and the environment. The agent can be viewed as the main processing component and the environment represents the domain the agent tries to solve. At any arbitrary timestep t, the environment passes the current state S_t and reward R_t to the agent, which in turn executes an action A_t in the environment. Agents use various *algorithms* to determine the best possible action A_t at state S_t. In our study, we primarily considered *Q-learning* (QL) [23] and *Policy*

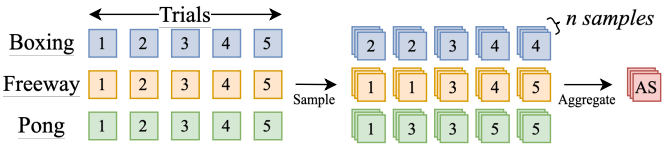

Fig. 1: Applying SBCI to all trials from a configuration.

Gradient (PG) [24] based algorithms as they were commonly implemented by libraries.

b) Stratified bootstrap confidence intervals: Stratified Bootstrap Confidence Intervals (SBCI) [25] is a systematic and standardized way of reporting performance-based metrics, like an agent's mean reward, in DRL. SBCI involves bootstrapping the results of a few similarly configured trials to create a distribution that represents the true result distribution for that particular configuration more accurately, as illustrated in Figure 1. All trials (left) are (1) stratified bootstrapped (sampling with replacement, with equal proportions per environment) to form a *sample* (middle), and (2) subsequently aggregated according to a metric (*e.g.,* mean) to form an *aggregated sample* (AS). After *multiple* aggregated samples are accumulated, a distribution is then formed with confidence guarantees. SBCI is particularly useful when testing an algorithm over a large suite of environments since it significantly reduces the number of trials needed per environment for reliable estimates.

III. Motivation

In this section, we discuss the literature review and the pilot study that motivated this paper.

A. Literature Review

Although we were aware of studies that made the assumption of interchangeable algorithm implementations [15]–[18], we decided to conduct a *systematic* and *reproducible* literature review to properly justify this assumption.

a) Methodology: We reviewed recently accepted papers from two popular AI conferences—the conference on *Neural Information Processing Systems* (NeurIPS) and the *International Conference on Machine Learning* (ICML), both of the year 2023. Since both conferences combined had more than 5K accepted papers (3,584 for NeurIPS and 1,865 for ICML), we used the conference data provided by Paper Copilot [26] and a Python library named PDFPlumber [27] to automatically download the accepted papers and filter them by searching for keywords. We selected these keywords based on *two assumption scenarios* that primarily occurred in the studies we were already aware of; (1) when studies use an implementation that was not the original [15], [16] or (2) when studies alternate between two different implementations of the *same* algorithm in their experiments [17], [18], with the latter being a manifestation of the former. Thus, to narrow down our literature to review, we searched for papers which had the key phrase "Reinforcement Learning" in their title as well as references to at least two popular DRL libraries (see Table I for the list of popular DRL libraries). The keywords used to search for popular DRL libraries were "RLlib, Baselines3, Tianshou, CleanRL, and Baselines". For completeness, all searches were done in a case-insensitive manner.

b) Analysis: In total, we identified ten papers from NeurIPS, six papers from ICML, and seven papers from other conferences under the assumption of interchangeable algorithm implementations. Examples of studies conforming to the first aforementioned scenario are the works by Beukman et al. [28], Chiappa et al. [29], and Gerstgrasser et al. [30], where alternate implementations were used over the original for the SAC [31], PPO, and DQN algorithms respectively. Examples of studies conforming to the second aforementioned scenario are the works by Raghunath et al. [17] and Wolk et al. [18]. Raghunath et al. [17] integrated their study with the SAC algorithm in a single-agent setting as well as a multi-agent setting to demonstrate effectiveness. However, Raghunath et al. used different SAC implementations for both settings, assuming that they were consistent and thus, interchangeable. Given that their experiments show marginal differences between both single-agent and multi-agent settings [17, Figure 2], if the SAC implementations were *not* interchangeable, a change in SAC implementation could easily *flip* the experimental outcomes. Similarly, Wolk et al. [18] integrated their study with the PPO algorithm in multiple settings. However, Wolk et al. used different PPO implementations for some settings, assuming they were interchangeable. This potentially invalidates their comparisons, especially for those with marginal differences [18, Table 2].

c) Key takeaways: With numerous studies found to be under the assumption of interchangeable algorithm implementations, the notion that researchers—as well as practitioners—assume interchangeability is well-justified and potentially warrants study re-examinations if implementations were found to *not* be interchangeable.

B. Pilot Study

We conducted a pilot study to attain an *initial* assessment on the extent and effects of implementation inconsistencies before conducting a more thorough comparison on a larger scale as DRL experiments can often be computationally expensive.

a) Methodology: We trained agents from five DQN implementations (with the same hyperparameters) to solve three games from the Atari 2600 benchmark [32], as illustrated in Figure 2. Since there were installation issues with the original DQN implementation, and the DQN's authors subsequently vouched for Dopamine's implementation as an accurate alternative in a GitHub issue, we used Dopamine's DQN implementation as the baseline.[1] Moreover, we also included our own DQN implementation (*i.e.,* Ours).

b) Analysis: We observed stark discrepancies in the training curves, with the majority of implementations performing suboptimally when compared to Dopamine, with either low final mean rewards or high variance among their trials (*i.e.,* elongated shaded regions). To account for these discrepancies,

[1]All links to issues and pull requests are included in the source code.

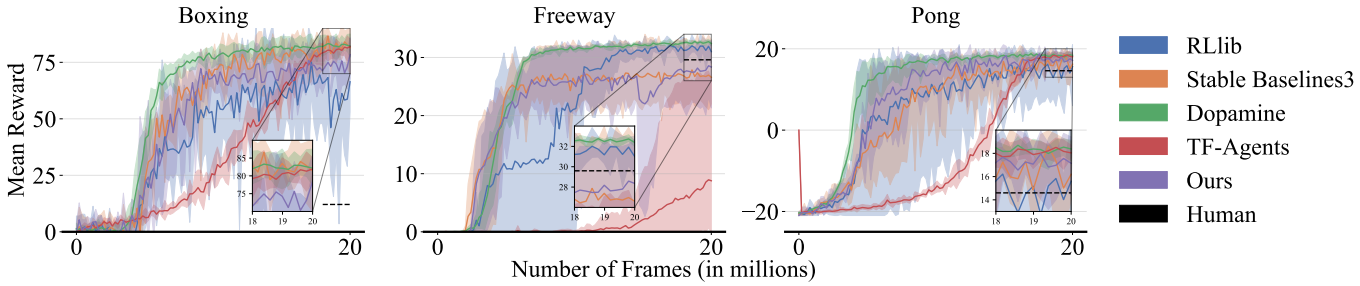

Fig. 2: Training curves from five DQN implementations where the y-axis represents the mean in-game reward while the x-axis represents the number of in-game frames that have passed. Five agents were trained for each (implementation, game) permutation and the training curves were aggregated to display the mean, minimum, and maximum within the shaded regions. The mean reward attained by a professional human tester is also shown in black to gauge superhuman or subhuman capabilities.

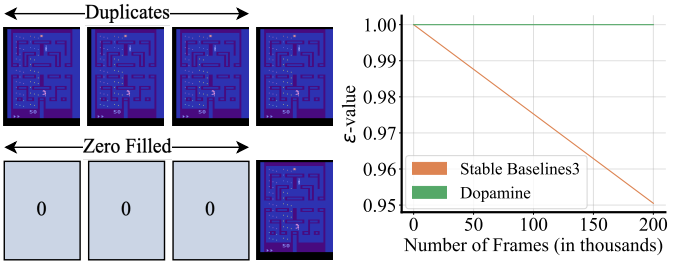

Fig. 3: Frame stacking approaches taken by Stable Baselines3 (top-left) and Dopamine (bottom-left), as well as a comparison of their implementation of the ϵ-greedy policy (right).

we investigated code-level inconsistencies. Firstly, we found that Stable Baselines3 [33] and Dopamine both differ in how they form the stack of frames as input to the Q-network, with the former duplicating frames to account for missing frames and the latter filling the missing frames with zeros instead, as illustrated in Figure 3 (left). Secondly and more importantly, we noticed that they also differ in how they decay certain hyperparameters, like the ϵ-value for the ϵ-greedy policy—a strategy where the ϵ-value (decayed with time) encourages an agent's exploration in an environment. As shown in Figure 3 (right), Stable Baselines3 starts the decay immediately, regardless of how many frames have passed, while Dopamine waits and only starts the decay much later on. As this directly impacts how an agent behaves, it could explain the discrepancy, in terms of curve steepness, between the two implementations.

c) Key takeaways: The discovery of efficacy discrepancies as well as major code-level inconsistencies among implementations of an already mature algorithm suggests the conjecture that implementations are *not* interchangeable and warrants a study of larger scale, with less mature algorithms and more environments.

IV. Methodology

In this section, we discuss the approach we adopted to address the three RQs. In particular, (1) how we selected, compared, and debugged implementations, (2) how we replicated a study, and (3) the experimental settings.

TABLE I: DRL Libraries

DRL Library	$\frac{1}{2}$[1]	DQN	PPO	Stars (K)	Last Commit
RLlib	✓	✓	✓	31.6[2]	2024
Dopamine	✓	✓	✗	10.4	2024
Stable Baselines3	✓	✓	✓	8.2	2024
Tianshou	✓	✓	✓	7.5	2024
CleanRL	✓	✓	✓	4.6	2024
OpenSpiel	✗	✓	✓	4.1	2024
ReAgent	✓	✓	✓	3.5	2024
ElegantRL	✓	✓	✓	3.5	2024
Acme	✓	✓	✓	3.4	2024
Tensorforce[3]	✓	✓	✓	3.3	2024
TF-Agents	✓	✓	✓	2.7	2024
TorchRL	✓	✓	✓	2.0	2024
Coach[3]	✓	✓	✓	2.3	2022
Tonic RL	✓	✗	✓	0.39	2021
Catalyst-RL	✓	✓	✓	0.046	2021
Baselines	✓	✓	✓	15.4	2020
Spinning Up[4]	✓	✗	✓	9.7	2020
Keras-RL	✓	✓	✗	5.5	2019

[1] At least half of the algorithms implemented are QL or PG-based.
[2] With respect to the entire Ray codebase and not just RLlib.
[3] Will no longer be maintained.
[4] Does not support GPU computations.

A. Selecting Algorithms and Implementations

Due to significant hardware requirements, testing and comparing all algorithms with their respective implementations would have been infeasible. Thus, in order to assess the prevalence of discrepancies between different implementations of the same algorithm in RQ1, we first determined which algorithms and implementations were representative of the DRL domain. Table I depicts a list of popular DRL libraries, sorted firstly by last commit date, and secondly by GitHub stars. Furthermore, we investigated the type of algorithms that were commonly implemented and determined that (1) QL or PG-based algorithms were the most common, and (2) DQN and PPO were the most common QL and PG-based algorithms respectively. Subsequently, we narrowed our scope to selecting just one algorithm, as it then allowed us to conduct a *large-*

scale comparison across more implementations with a large environment suite.

We chose to test and compare PPO implementations over DQN implementations for RQ1 due to their low memory costs. Memory was a primary concern because it determines how many trials we were able to run concurrently on our GPU servers. Since the memory costs for DQN are significantly higher than that of PPO (*e.g.,* 1M states vs 1K states in the Atari 2600 benchmark [9], [20]), we chose to test PPO implementations instead. To put this in a better perspective, we could run eight concurrent PPO trials compared to two concurrent DQN trials on our GPU servers. Moreover, we limited our selection of PPO implementations to just five implementations, so that we could incorporate a larger suite of environments in our tests. Thus, we selected the implementation by Baselines since it was the original PPO implementation and subsequently selected implementations from the four most popular actively maintained libraries for RQ1, to wit, RLlib, Stable Baselines3, Tianshou [34], and CleanRL [35]. It would be a strong indication that libraries are inconsistent with their implementations of other algorithms if they were already inconsistent with a widely used algorithm, like PPO.

B. Comparing DRL Performance

Differential testing compares the outputs of two systems when given the same input. Should the outputs be the same, the systems are considered to be correct for the given input. Conversely, should the outputs be different, the systems are considered to have differing behaviour. Differential testing is primarily used as a black-box testing technique and thus, widely applicable for systems with neural networks [36]–[40]. However, the same differential testing methodology used by the aforementioned DL studies cannot be directly applied to DRL without first addressing the domain's stochasticity.

a) Differential testing in DRL: The core challenge of applying differential testing to DRL systems is the inherent stochasticity present in the systems. A given input might not consistently produce the same output, as observed in the pilot study, in Figure 2. We address this uncertainty by using SBCI to attain accurate and reliable estimates. In particular, when testing PPO implementations in RQ1, the *inputs* were PPO's hyperparameters—an algorithm's configuration, manually set to be consistent with the original publication, across all tested implementations. To compare fairly with other PPO implementations, utilizing SBCI, the *outputs* of all trials from a PPO implementation were stratified bootstrapped with respect to multiple *aggregation metrics*, to evaluate different aspects of efficacy. This results in *point estimates* with *confidence bands* for all aggregation metrics used, for all PPO implementations tested, allowing for a more accurate and reliable comparison.

b) Environments: Since an agent's performance is dependent on the difficulty of an environment [32], to further increase the fairness of our comparisons in RQ1, we used a large and diverse suite of environments from the Atari 2600 benchmark (56 environments) when testing the PPO implementations—the same environments commonly used to benchmark new algorithms [8], [9], [14], [20], [41], [42]. This was done to ensure that all implementations were tested in as many situations as possible to better compare their agents' generalization capabilities—to perform in any environment.

c) Human normalized score: Prior to the application of any statistical techniques post-training, it is common to first normalize the outputs (*i.e.,* mean rewards) of trials from the Atari 2600 benchmark suite with respect to the mean rewards attained by a professional human tester, so as to properly gauge superhuman performance in games [14], [20], [41], [42]. Thus, to obtain a single human normalized score representing the efficacy of a PPO trial in RQ1, we first took the mean reward attained from the last 100 training episodes instead of just the reward from the last training episode for a more accurate estimate of an agent's efficacy [9], [43]. Subsequently, we applied min-max normalization to the mean reward,

$$Score = \frac{MeanReward_{100} - RandomPlay}{HumanPlay - RandomPlay} \tag{1}$$

where min and max represent the reward attained from random and human play respectively, referenced from a previous study [42]. The agent is superhuman if $Score > 1$.

d) Statistical techniques: In this study, it is essential that we draw statistically-sound conclusions about any potential discrepancies among the implementations, particularly when dealing with stochastic algorithms. To this end, aside from statistical tests like ANOVA [44], we used a variety of state-of-the-art techniques specifically tailored for DRL.

Firstly, we incorporated an environment-wise one-way ANOVA to compare the effect of PPO implementation on $MeanReward_{100}$. With a null hypothesis of equal implementation means, rejecting it (*i.e.,* p-value < 0.05) translates to a statistically significant difference in means. However, (1) a one-way ANOVA does not pinpoint outliers, (2) with large amounts of data, minor effects can easily cause statistically significant differences, and (3) statistical insignificance does not imply absence of effect [25], [45]. Thus, we decided to include additional techniques for broader perspectives.

Subsequently, to further compare the PPO implementations, we applied SBCI to their scores with respect to two different aggregation metrics [25]; (1) fraction of trials with $Score > \tau$, and (2) probability of improvement (POI). The first metric results in a *performance profile* across all of an implementation's trials, to gauge the number of trials that achieved superhuman game performance (*i.e.,* when $\tau = 1$). The second metric uses more direct, one-to-one *pairwise comparisons* with the Mann-Whitney U-statistic [46], that is,

$$\begin{aligned} P(X_m > Y_m) &= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} S(x_{m,i}, y_{m,j}), \\ where\ \ S(x,y) &= \begin{cases} 1, & \text{if } y < x, \\ \frac{1}{2}, & \text{if } y = x, \\ 0, & \text{if } y > x. \end{cases} \end{aligned} \tag{2}$$

Here, we directly compute the POI of implementation X over implementation Y with N trials each, on environment m. The

primary assumption is that for X to be better than Y, X has to outperform Y *sufficiently often*. The final POI of X over Y, $P(X > Y)$, is subsequently attained by calculating the POI for all environments $\frac{1}{M}\sum_{m=1}^{M} P(X_m > Y_m)$. Post-SBCI, the point estimates and confidence bands attained from this metric are further analyzed with the Neyman-Pearson testing criterion [47], [48] for statistical significance and meaningfulness. Statistical significance where $P(X > Y) > 0.5 \ \wedge \ 0.5 \notin CI$ rules out the effect of noise on the point estimates. On the other hand, statistical meaningfulness where $CI_{upper} > 0.75$ ensures that X outperforms Y often enough. X is considered to be *better* than Y if both aforementioned criteria are met. Thus, the POI metric directly measures the likelihood that an implementation X outperforms implementation Y on a random environment, complementing the performance profiles.

C. Debugging Discrepancies

To locate the root cause of discrepancies found in RQ1 for RQ2, we adopted a *best-effort* debugging approach similar to that of the pilot study. Moreover, we focused on the high-performing PPO implementations as they likely differed from code-level inconsistencies rather than critical bugs. In particular, we manually inspected the PPO implementations' source codes to identify any inconsistencies that could account for the discrepancies found. Upon finding any inconsistencies, we (1) corrected them to be consistent across implementations, (2) re-tested the implementations, and (3) re-compared the implementations to determine if the discrepancies were still present. Although time-consuming as this required us to build an in-depth understanding of the source codes, it was necessary in order to ascertain that the inconsistencies were indeed the cause of the discrepancies found. Lastly, we also informed the implementations' developers, via GitHub issues, of the unresolved discrepancies, as a final attempt to address them.

D. Replicating a Study

In order to determine if the assumption of interchangeable implementations could alter the outcomes of an experiment in RQ3, we replicated experiments from a study that made the assumption. Islam et al. [16] opted to use an alternate implementation of the DDPG algorithm over the original [22] and recommended ideal hyperparameter values for the DDPG algorithm, assuming that the DDPG implementations were interchangeable. We chose this specific study because (1) many works based on the study's results, as its more than 300 citations show, (2) it adhered to the current best practices by providing detailed descriptions of the hyperparameter values and implementations used, and (3) it was simple. In particular, the study investigated the best value for a few commonly used hyperparameters, in terms of efficacy. The study tested different values for each hyperparameter investigated with DDPG on the HalfCheetah environment from the MuJoCo benchmark suite [49]. We replicated the experiments involving the network architecture, reward scale, and batch size hyperparameters, but with a different DDPG implementation—Stable Baselines3's DDPG implementation—to investigate if using a different implementation was significant enough to *flip* experiment outcomes. Furthermore, we used the same differential testing methodology as in RQ1 (and RQ2), for accurate and reliable comparisons.

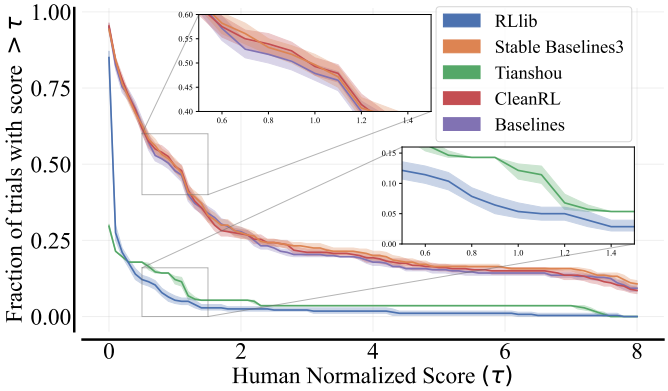

Fig. 4: Performance profiles for the five PPO implementations tested across 56 environments, where the shaded regions indicate pointwise 95% confidence bands based on SBCI.

E. Experimental Settings

We conducted our experiments concurrently on three GPU servers running the Ubuntu LTS operating system. The first server had three NVIDIA RTX A4000 GPUs with an AMD Ryzen Threadripper 3970X CPU. The second server had four NVIDIA RTX 2070 SUPER GPUs with an Intel Core i9-10900X CPU. The last server had one NVIDIA RTX 3090 GPU with an AMD Ryzen 9 5950X CPU. As we did not compare in terms of training speed, the trials were comparable across servers—DRL libraries use either TensorFlow or PyTorch, which conforms to 32-bit floating point precision by default. Furthermore, when configuring the PPO implementations for RQ1 (and RQ2), we used a hyperparameter set similar to the one used by the original implementation [9], with the only difference being that we excluded LSTM layers in the neural network, to speed up training [50]. We opted for the speed-up since this is a large-scale comparison-focused study, amounting to approximately 10K GPU hours. When configuring Stable Baselines3's DDPG implementation for RQ3, we followed the hyperparameter set used by the replicated study [16]. We trained five agents for each (1) (implementation, environment) permutation for RQ1 (and RQ2) and (2) (hyperparameter, value) permutation for RQ3. We trained five agents (*i.e.,* five trials) for each permutation as it was sufficient for reliable SBCI estimates with the Atari benchmark [25].

V. Results

In this section, we present the outcomes from our experiments addressing RQ1, RQ2, and RQ3.

A. Prevalence of Implementation Discrepancies

The performance profiles in Figure 4 demonstrate stark discrepancies between the five PPO implementations tested.

TABLE II: Discrepancies Among The High-Performing PPO Implementations

Game	Mean Reward Over 5 Trials				One-way ANOVA[1]					
	Stable Baselines3	CleanRL	Baselines	Baselines108	F-statistic[2]	p-value[2]	Reject[2]	F-statistic[3]	p-value[3]	Reject[3]
Atlantis	881191.6	946388.0	2088141.2	848217.6	262.634	1.241e-10	**Yes**	2.925	9.229e-02	**No**
ChopperCommand	5519.0	6056.8	841.4	913.6	157.884	2.408e-09	**Yes**	152.909	2.897e-09	Yes
DoubleDunk	-2.7	-3.4	-13.4	-3.0	956.997	5.850e-14	**Yes**	2.713	1.066e-01	**No**
Gopher	1521.1	4946.4	4752.1	5086.4	8.636	4.747e-03	**Yes**	8.189	5.718e-03	Yes
Krull	10369.7	9589.3	9535.5	9653.7	17.274	2.936e-04	**Yes**	4.453	3.577e-02	Yes
Robotank	19.8	9.5	13.9	12.7	5.365	2.165e-02	**Yes**	6.366	1.305e-02	Yes
Tennis	-4.2	-4.2	-14.2	-3.9	7.101	9.229e-03	**Yes**	0.004	9.959e-01	**No**
VideoPinball	37962.8	55677.5	48812.4	72580.8	5.875	1.664e-02	**Yes**	4.793	2.951e-02	Yes
Zaxxon	10241.2	5548.4	6024.6	6685.6	6.263	1.372e-02	**Yes**	7.351	8.238e-03	Yes

[1] Reject null hypothesis of equal implementation means if p-value < 0.05.
[2] With respect to Stable Baselines3, CleanRL, and Baselines.
[3] With respect to Stable Baselines3, CleanRL, and Baselines108—108K frames per episode.

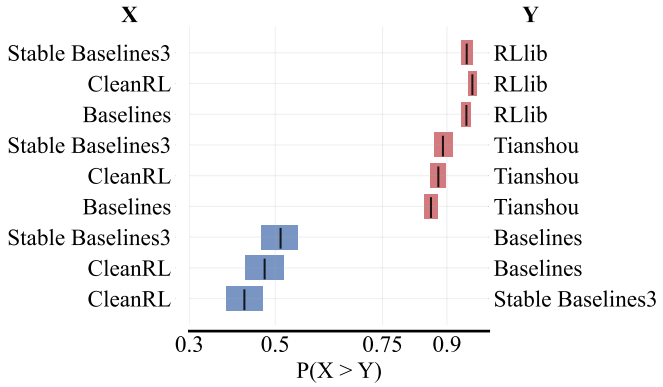

Fig. 5: Pairwise POI for the five PPO implementations tested across 56 environments, where vertical stripes represent point estimates, and shaded regions indicate 95% confidence bands based on SBCI. X is considered to be *better* than Y if the point estimate is both statistically significant and meaningful. Comparisons where X is better than Y are indicated in red.

Specifically, we can see two groups with similar performance; (1) Stable Baselines3, CleanRL, and Baselines where 50% of their trials attained superhuman performance (*i.e.,* $\tau > 1$), and (2) RLlib and Tianshou where less than 15% of their trials attained superhuman performance. For conciseness, we subsequently refer to these two groups as high-performing and low-performing groups respectively. Although it cannot yet be inferred that the high-performing implementations are, in general, superior in terms of performance.

a) Probability of improvement between implementations: Figure 5 shows that the high-performing group is objectively better than the low-performing group. In particular, the pairwise POI between the two groups is both statistically significant, where $P(X > Y) > 0.5 \ \wedge \ 0.5 \notin CI$, and meaningful, where $CI_{upper} > 0.75$. From the pairwise comparisons, it is also observed that Stable Baselines3, CleanRL, and Baselines are on par with each other, with the pairwise comparisons among themselves being neither statistically significant nor meaningful. However, as shown in Table II, when considering individual environments, we found statistically significant discrepancies. We later determined these discrepancies to be caused by implementation inconsistencies in RQ2.

b) Per-environment discrepancies: The one-way ANOVA in Table II with respect to Baselines (not the 108 variant) shows that the PPO implementations from the high-performing group were found to significantly differ in nine environments, contrasting the above-discussed aggregated comparisons. This discrepancy can also be observed in the training curves, shown in Figure 6, with Atlantis and DoubleDunk. Specifically, the curves from Stable Baselines3 and CleanRL noticeably differ from Baselines' curves.

The PPO implementations from the low-performing group were observed to attain low mean rewards in the majority of environments, similar to the pattern observed in Assault and Breakout. Tianshou could, however, solve some *simple* environments, like Pong and Tennis. These environments, however, have low score caps that are easily reached by agents [51], and therefore, not a strong indication of an implementation's predictive power. Lastly, all five PPO implementations were unable to solve some environments like MontezumaRevenge and PrivateEye. This was, however, expected because these environments have sparse rewards [32], making them especially *hard* in the context of DRL, where agents use rewards as feedback for previously executed actions.

c) Key takeaways: All of the aforementioned observations point strongly towards code-level inconsistencies and bugs between the tested PPO implementations. In particular, environment-dependent inconsistencies between the high-performing implementations, and critical bugs with the low-performing implementations. This is surprising, considering that (1) PPO is a common algorithm, (2) the implementations were from established libraries, and (3) Atari 2600, one of the most widely known benchmarks, was used.

Thus, researchers should exercise caution when interchanging implementations of the PPO algorithm as their experiments might yield different outcomes, depending on the implementation and environment used. Furthermore, since code-level inconsistencies are observed to be a recurring pattern (also with DQN implementations), this now raises concerns about the reliability of the implementations of other less mature

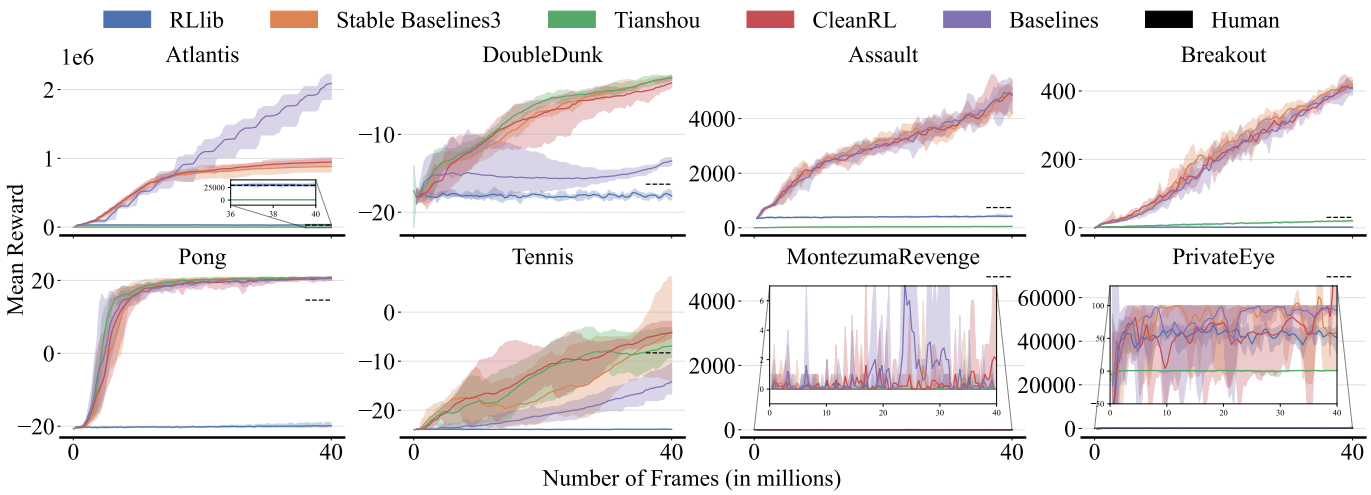

Fig. 6: Training curves from five PPO implementations. Similar to DQN, five agents were trained for each (implementation, environment) permutation and the training curves were aggregated to display the mean, minimum, and maximum within the shaded regions. The mean reward attained by a professional human tester is also shown in black to gauge superhuman or subhuman capabilities. For brevity, only eight representative environments (out of 56) are displayed.

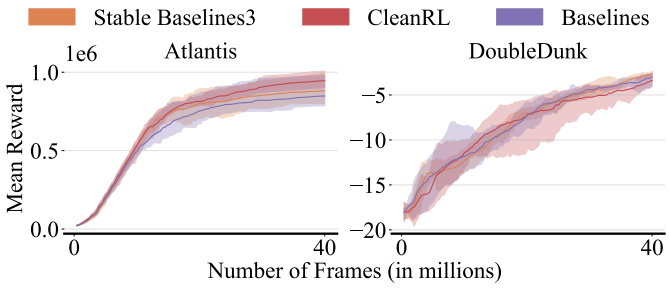

Fig. 7: Consistent training curves among the high-performing PPO implementations after explicitly configuring the maximum frames per episode to be 108K. Like RQ1, each (implementation, environment) permutation underwent five trials.

algorithms as well. Researchers should not take the reliability and interchangeability of implementations for granted.

RQ1 Summary: Discrepancies between different implementations of the same algorithm are prevalent. In particular, out of the five popular PPO implementations tested on 56 environments, three implementations attained superhuman performance for 50% of their trials while the other two implementations only attained superhuman performance for less than 15% of their trials. Moreover, among the high-performing implementations, statistically significant performance differences were also observed in nine environments.

B. Reasons for Discrepancies

In our manual analysis, we found three code-level inconsistencies among the high-performing PPO implementations, one of which explained the performance differences. In particular, the inconsistency with the number of frames per episode influenced the performance while the inconsistencies with gradient clipping and averaging did not.

a) Frames per episode: Firstly, Baselines used the Gym library [52] for the Atari 2600 benchmark, which defaulted to *more than* 108K frames per episode while Stable Baselines3 and CleanRL used the Gymnasium library [53], which defaulted *strictly* to 108K frames per episode. This was one of the reasons for the discrepancies observed in RQ1. Since the frames per episode were higher in Baselines, their agents spent more time in an environment before it gets reset—directly affecting the training process. After correcting this by configuring `max_episode_steps` to be 27K (108K frames) in Baselines' environment preprocessing function `make_atari`, the implementations no longer significantly differed in three environments (out of nine), as seen in Table II. Furthermore, the training curves among the implementations were also observed to be more consistent when compared to those of RQ1, as shown in Figure 7.

b) Gradient clipping: Secondly, the implementations differed in how they clipped gradients—commonly used to stabilize updates in a neural network and is dependent on the DL library used. In particular, Stable Baselines3 and CleanRL used `clip_grad_norm` from the PyTorch library to clip gradients while Baselines used `clip_by_global_norm` from the TensorFlow library. These two functions clip the gradients differently. To determine if this clipping inconsistency caused the discrepancies observed in RQ1, we (1) disabled gradient clipping across all high-performing implementations, and (2) re-tested and re-compared the implementations. The performance without gradient clipping exhibited the same discrepancies as before and therefore, the clipping inconsistencies were not significant enough to have caused the discrepancies.

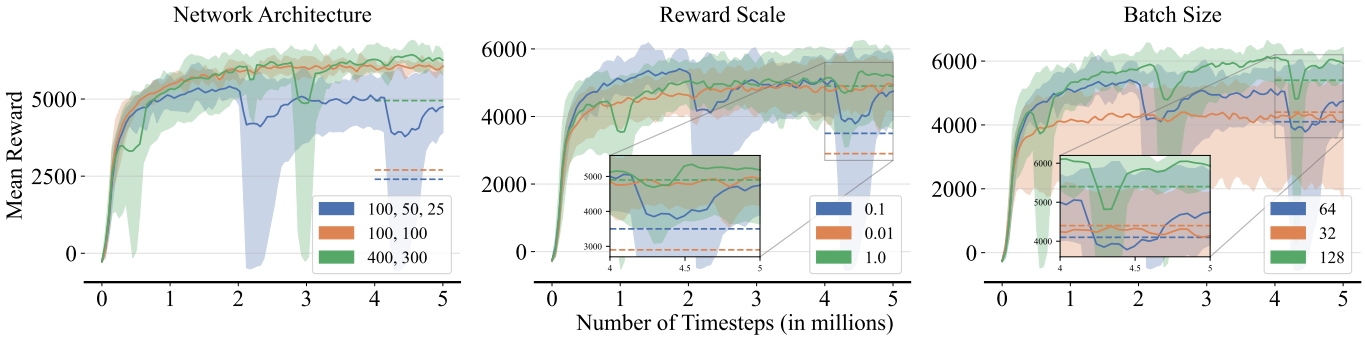

Fig. 8: Training curves for three DDPG hyperparameter experiments on the HalfCheetah environment. Similar to PPO, five agents were trained for each (hyperparameter, value) permutation and the training curves were aggregated to display the mean, minimum, and maximum within the shaded regions. The dotted lines represent the efficacy reported in the replicated study.

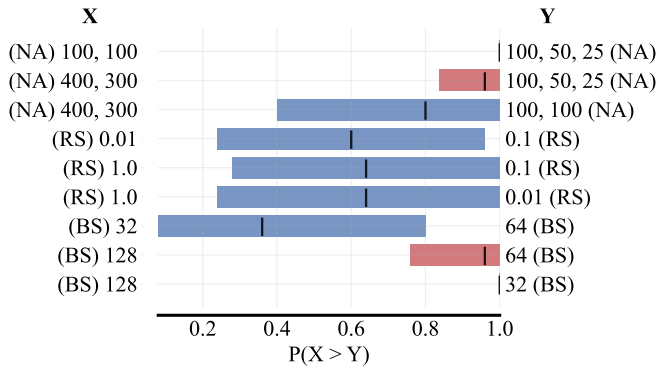

Fig. 9: Pairwise POI for three DDPG hyperparameter experiments, where vertical stripes represent point estimates, and shaded regions represent 95% confidence bands based on SBCI. X is considered to be *better* than Y if the point estimate is both statistically significant and meaningful. Comparisons where X is better than Y are indicated in red.

c) Gradient averaging: Lastly, Baselines also averaged gradients over multiple processes using OpenMPI [54] to make updates in the neural network more efficient, something not implemented by either Stable Baselines3 or CleanRL. As this would have affected the gradients in the neural network, we (1) disabled gradient averaging in Baselines, and (2) re-tested and re-compared the implementations. The performance without gradient averaging exhibited the same discrepancies as before and therefore, the averaging inconsistencies were not significant enough to have caused the discrepancies.

d) Key takeaways: We found multiple implementation inconsistencies, one of which had a significant impact on performance. The frames per episode inconsistency is an example of a difference that has a major impact on performance and that even the original implementation can be inconsistent. In fact, there are six environments in Table II that still differ—hinting at more inconsistencies among the high-performing implementations. The three inconsistencies observed can be classified as either API-based where the issue stems from the use of an external API (like frames per episode or gradient clipping), or logic-based where the issue lies internally (like gradient averaging)—types of inconsistencies also commonly found among AI implementations [55], [56]. Moreover, we do not believe that these inconsistencies are in any way intentional, but rather, oversights stemming from a domain of high complexity, a phenomenon commonly observed with DL implementations [55]. This reaffirms the notion that it is *not* necessarily the case that DRL implementations are consistent and interchangeable because of the *high* risk of inconsistencies associated with the DRL paradigm—being a combination of both DL and RL.

RQ2 Summary: Implementation discrepancies occur primarily because of code-level inconsistencies between the implementations. In particular, we found three inconsistencies among the high-performing PPO implementations. Furthermore, correcting these inconsistencies solved discrepancies among the high-performing implementations in three environments.

C. Altering Experiment Outcomes

The study we replicated explored the best values, in terms of efficacy, for commonly used hyperparameters with DDPG on the HalfCheetah environment. Using a different DDPG implementation, we replicated the experiments for three hyperparameters, to wit, network architecture (NA), reward scale (RS), and batch size (BS). The training curves in Figure 8 and their respective pairwise POI in Figure 9 contrast those originally reported by the replicated study.

a) Network architecture: Firstly, for the experiment on network architecture, it can be seen in the training curves that the (400, 300) architecture variant does *not* dominate the other two variants as previously reported in the study (*i.e.,* the dotted lines), but instead, is on par with the (100, 100) variant, and that both of them dominate the (100, 50, 25) variant. Their respective pairwise POI reaffirms this, where (1) both (400, 300) and (100, 100) are better than (100, 50, 25), and (2) (400, 300) is not better than (100, 100).

b) Reward scale: Secondly, for the experiment on reward scale, it can be seen in the training curves that a scale of 1 (*i.e.,* no scaling) does *not* dominate the other two variants as previously reported in the study, but instead, is on par with both of them. Their respective pairwise POI reaffirms this, where neither variant was better than the other.

c) Batch size: Lastly, for the experiment on batch size, the training curves show that the 128 variant dominates the other two variants, similar to what was previously reported in the study. Their pairwise POI reaffirms this, where (1) 128 is better than both 64 and 32, and (2) 32 is not better than 64.

d) Implications: We have demonstrated that the empirical outcomes originally reported by the study, although not incorrect, can differ based on the algorithm implementation used. Given that the study aimed to guide researchers towards ideal hyperparameter configurations, researchers could build on the conclusions of the study and *not* see the claimed advantages due to these variances in outcomes.

e) Key takeaways: This study was just *one of the many* studies that we could have replicated and we selected it because it adheres to the current best evaluation practices. However, the fact that using a different implementation was significant enough to alter experiment outcomes has serious implications on existing DRL research. In particular, with research assuming that implementations are interchangeable [15]–[18], possibly warranting follow-up studies to replicate them and test whether their claims can be reproduced.

RQ3 Summary: The assumption of interchangeable implementations was found to be significant enough to be able to alter the outcomes of an experiment. In particular, two out of three experiment outcomes we replicated with a different DDPG implementation differed from those of the original study.

VI. Discussion

In this section we discuss the (1) key takeaways, (2) actionable recommendations, and (3) issues and pull requests we filed for the discrepancies and inconsistencies found.

a) Implementations are not interchangeable: In this study, we have shown that the assumption of interchangeable DRL implementations is (1) common among studies, (2) detrimental to a study's internal validity, potentially *flipping* experiment outcomes, and thus, (3) *mistaken*. Hence, we recommend for replicability studies to test whether the claims of notable studies under this assumption can be reproduced. Furthermore, we recommend for the replicability studies to adopt the differential testing methodology used in this study to increase the fairness of their comparisons.

b) Implementation inconsistencies exist: In RQ1 and RQ2, we have shown that implementation discrepancies are prevalent and that code-level inconsistencies cause these discrepancies. Two out of three inconsistencies found were API-based, suggesting that the complexity of DRL implementations, including their hyperparameters and stochasticity, make it difficult for unit testing to eliminate potential issues—something also observed in the DL domain [55]. Thus, we believe that individually checking implementations will not solve the inconsistencies in the long run. The inconsistencies can instead, be addressed with a more sustainable approach, at either the implementation or usage stage.

Firstly, for the implementation stage, we recommend that developers of DRL libraries proactively compare against other implementations using the differential testing methodology proposed in this study. This differential testing methodology can be easily automated with CI/CD and is already incorporated in other domains to address implementation inconsistencies as well [57], [58]. In fact, in response to our GitHub issue, Tianshou indicated that they already had plans for this. Secondly, for the usage stage, we recommend that developers of DRL libraries explicitly document code-level inconsistencies for their implementations. Researchers should conduct their experiments with this in mind—acknowledging that these inconsistencies exist and avoiding the risk of basing their conclusions on them. We believe that this paves the way for more reliable and consistent implementations.

c) Testing granularities: Although the differential testing methodology used in this study is end-to-end (*i.e.,* testing the entire algorithm), we would like to highlight that differential testing of individual functions or components is feasible as well. However, one significant drawback in doing so is that only a subset of functions can be tested, as noted by Herbold and Tunkel [59] when they differential tested *Machine Learning* (ML) libraries. This is because of differences between the hyperparameters and implementations of the functions. For example, some DRL libraries might expose only a subset of hyperparameters in their functions, or even combine functions together, making it difficult to differential test them. Thus, we opted to test end-to-end for completeness. Nonetheless, functional testing has its advantages as well; (1) computational costs can be reduced by testing only what needs to be tested, and (2) inconsistencies can be localized more easily by only inspecting the functions that were tested. Thus, we recommend for researchers and practitioners to test end-to-end when prioritizing completeness and at a functional level when prioritizing computational costs and inconsistency localization.

d) Large test suites are preferred: A test suite's size is correlated with its effectiveness [60] and should cover as many situations as possible to increase its effectiveness [61]. Had we instead opted for any of the *optimal* Atari environment subsets proposed by Atari-5 [51], discrepancies among the high-performing PPO implementations with six environments in Table II would still have gone *undetected*. This underscores the importance of using a large environment suite for comparative studies in the DRL domain. This is even more applicable to studies on the extent of implementation discrepancies, where existing studies in this area only used a small environment suite for their experiments [19], [62]. Thus, we recommend for DRL researchers and practitioners to prioritize using a large environment suite whenever possible, with SBCI. The additional computation costs can be significantly reduced

by using SBCI to effectively minimize the number of trials required for accurate and reliable estimates.

e) Issues and pull requests: In total, we filed six issues and one pull request on GitHub; (1) five issues and one pull request for the discrepancies and inconsistencies found in RQ1 and RQ2—one issue for each DRL library tested and one pull request for Baselines' *frames per episode* inconsistency, and (2) one issue for a discrepancy we found with RLlib's SAC [63] implementation during our preliminary experiments. In response to our issues, the developers from Stable Baselines3 and CleanRL uncovered two additional inconsistencies regarding timeouts and value clipping that could account for the discrepancies among the remaining six environments in Table II. The developers from Tianshou actively investigated and confirmed that the cause for the discrepancies was because of a misleading code example in their repository while the developers from Baselines and RLlib have yet to acknowledge our issues or pull request. RLlib's developers, however, acknowledged our issue regarding the SAC discrepancy, which was later confirmed to be because of an inconsistency.

VII. Threats To Validity

Here, we detail the steps taken to assess and mitigate the most important threats to validity.

a) Internal validity: Firstly, one concern was with incorrectly configuring the implementations. Since libraries aim to implement algorithms that are both logically the same and compatible with their existing architecture (*e.g.,* by inheriting from existing classes core to the library), they end up having different naming conventions and function signatures. For example, all of the PPO code for CleanRL is contained within a single file with minimal functions, while the PPO code for RLlib spans multiple files and functions. This diversity among implementations is not specific to DRL and is also prevalent in other domains as well, like ML [59].

To reduce the likelihood of a configuration oversight, we closely followed the code examples provided by the implementations, modifying only where necessary. Furthermore, to ascertain that the discrepancies found were not from a configuration oversight, we adopted a *best effort* approach when debugging them. In particular, we (1) manually re-inspected the code for inconsistencies that could account for the discrepancies, (2) re-tested and re-compared when an inconsistency was found, and (3) contacted the implementations' developers when we could not fix the discrepancy, as a final attempt to address it. Moreover, we focused on the high-performing implementations as their discrepancies were more likely to stem from actual implementation inconsistencies, rather than configuration oversights or critical bugs. Nonetheless, we still informed developers from all DRL libraries of the discrepancies found. Lastly, we are confident that any configuration oversights would not affect the key takeaways of this study, as they could at most affect a small subset of the results——the discrepancies without known inconsistencies from RLlib's PPO implementation.

b) External validity: Secondly, one other concern is that only implementations of two algorithms were tested. We selected DQN and PPO because they are both common (implemented in 16 out of the 18 DRL libraries inspected) and from different DRL paradigms (QL and PG). If discrepancies and code-level inconsistencies were prevalent among their implementations, there is a high likelihood that they are prevalent among implementations of other algorithms as well. For example, inconsistencies we discovered *a year ago* with RLlib's implementation of SAC with their 2.2.0 release that are still present in their current release—2.34.0. Furthermore, their experimental results could potentially be extrapolated to other similar algorithms in their respective paradigms as well, as demonstrated with DDPG in RQ3.

VIII. Related Work

In this section, we discuss existing related literature.

a) Implementation discrepancies in DRL: While there have been recent studies reporting implementation discrepancies in DRL, it was not their primary focus. One study focused on the nuances between two *different* algorithms [19] while another focused on how different values for hyperparameters influenced the outcome [62], similar to [16]—the study we replicated in RQ3. Building on the foundations laid by these studies, our work incorporates several additional methodologies to further deepen the understanding of implementation discrepancies; (1) *systematic* literature reviews and implementation selection, (2) *large* test suites, (3) DRL tailored *statistical* techniques, and (4) root cause *investigation*.

b) Differential testing in AI: There has been an abundance of studies using differential testing to identify and localize discrepancies in AI. In DL, the most common approaches include either fuzzing, mutation, or targeted searching to generate test cases for differential testing [36]–[40], and can range from applications such as generating adversarial inputs [64]–[66], testing DL implementations [67], [68], and large language model based fuzzers [69]–[72]. Moreover, there have also been comparative studies similar to this study, but instead, accessing DL and ML implementations [59], [73], [74]. Different from these studies, we now apply differential testing in the DRL domain to investigate the assumption of interchangeable DRL implementations.

IX. Conclusion

We conducted a large-scale testing-focused study to investigate the assumption of interchangeable DRL implementations. We observed significant discrepancies among the different implementations, later determined to be caused by code-level inconsistencies. We demonstrated that this assumption of implementation interchangeability was then significant enough to alter experimental outcomes. Lastly, we provided recommendations for DRL researchers and practitioners on how to reliably use, compare, and test DRL implementations. With the emerging field of integrating DRL with software testing [75]–[77], it is imperative to assess and improve the reliability of DRL implementations and studies, sooner rather than later.

ACKNOWLEDGMENTS

This research was supported by a Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant and the Shenzhen Science and Technology Program (No. KJZD20240903095700001).

REFERENCES

[1] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[2] A. M. Abdulazeez *et al.*, "A review on deep reinforcement learning for autonomous driving," *Indonesian Journal of Computer Science*, vol. 13, no. 3, 2024.

[3] K. A. ElDahshan, H. Farouk, and E. Mofreh, "Deep reinforcement learning based video games: A review," in *2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*. IEEE, 2022, pp. 302–309.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aar6404

[5] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. D. 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 andJonathan 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," 2019. [Online]. Available: https://arxiv.org/abs/1912.06680

[6] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou, "Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[7] X.-Y. Liu, H. Yang, J. Gao, and C. D. Wang, "FinRL: Deep reinforcement learning framework to automate trading in quantitative finance," *ACM International Conference on AI in Finance (ICAIF)*, 2021.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

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

[10] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "Rllib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3053–3062.

[11] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[12] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A Research Framework for Deep Reinforcement Learning," 2018. [Online]. Available: http://arxiv.org/abs/1812.06110

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[14] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[15] T. Zeng, X. Zhang, J. Duan, C. Yu, C. Wu, and X. Chen, "An offline-transfer-online framework for cloud-edge collaborative distributed reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[16] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," *Reproducibility in Machine Learning Workshop, ICML*, 2017.

[17] R. Raghunath, B. Peng, K.-L. Besser, and E. A. Jorswieck, "Reinforcement learning-based global programming for energy efficiency in multi-cell interference networks," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 5499–5504.

[18] M. Wolk, A. Applebaum, C. Dennler, P. Dwyer, M. Moskowitz, H. Nguyen, N. Nichols, N. Park, P. Rachwalski, F. Rau *et al.*, "Beyond cage: Investigating generalization of learned autonomous network defense policies," *Neural Information Processing Systems Workshop, NeurIPS*, 2022.

[19] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep rl: A case study on ppo and trpo," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=r1etN1rtPB

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] W. M. McKeeman, "Differential testing for software," *Digital Technical Journal*, vol. 10, no. 1, pp. 100–107, 1998.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *International Conference on Learning Representations*, 2016.

[23] C. J. C. H. Watkins, "Learning from delayed rewards," *PhD thesis, University of Cambridge*, 1989.

[24] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[25] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," *Advances in Neural Information Processing Systems*, 2021.

[26] J. Yang, "Paper copilot," https://papercopilot.com.

[27] J. Singer-Vine and The pdfplumber contributors, "pdfplumber," Aug. 2024. [Online]. Available: https://github.com/jsvine/pdfplumber

[28] M. Beukman, D. Jarvis, R. Klein, S. James, and B. Rosman, "Dynamics generalisation in reinforcement learning via adaptive context-aware policies," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[29] A. S. Chiappa, A. Marin Vargas, A. Huang, and A. Mathis, "Latent exploration for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[30] M. Gerstgrasser, T. Danino, and S. Keren, "Selectively sharing experiences improves multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[31] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[32] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, jun 2013.

[33] 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

[34] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, Y. Su, H. Su, and J. Zhu, "Tianshou: A highly modularized deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 23, no. 267, pp. 1–6, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1127.html

[35] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[36] Y. Deng, C. Yang, A. Wei, and L. Zhang, "Fuzzing deep-learning libraries via automated relational api inference," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 44–56.

[37] Q. Guo, X. Xie, Y. Li, X. Zhang, Y. Liu, X. Li, and C. Shen, "Audee: Automated testing for deep learning frameworks," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, 2020, pp. 486–498.

[38] A. Wei, Y. Deng, C. Yang, and L. Zhang, "Free lunch for testing: Fuzzing deep-learning libraries from open source," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 995–1007.

[39] X. Zhang, J. Liu, N. Sun, C. Fang, J. Liu, J. Wang, D. Chai, and Z. Chen, "Duo: Differential fuzzing for deep learning operators," *IEEE Transactions on Reliability*, vol. 70, no. 4, pp. 1671–1685, 2021.

[40] Y. Deng, C. S. Xia, C. Yang, S. D. Zhang, S. Yang, and L. Zhang, "Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3623343

[41] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[42] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[43] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[44] E. R. Girden, *ANOVA: Repeated measures*. Sage, 1992, no. 84.

[45] S. Greenland, S. J. Senn, K. J. Rothman, J. B. Carlin, C. Poole, S. N. Goodman, and D. G. Altman, "Statistical tests, p values, confidence intervals, and power: a guide to misinterpretations," *European journal of epidemiology*, vol. 31, no. 4, pp. 337–350, 2016.

[46] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[47] X. Bouthillier, P. Delaunay, M. Bronzi, A. Trofimov, B. Nichyporuk, J. Szeto, N. Mohammadi Sepahvand, E. Raff, K. Madan, V. Voleti *et al.*, "Accounting for variance in machine learning benchmarks," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 747–769, 2021.

[48] J. Neyman and E. S. Pearson, "On the use and interpretation of certain test criteria for purposes of statistical inference: Part i," *Biometrika*, pp. 175–240, 1928.

[49] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[50] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," in *ICLR Blog Track*, 2022, https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[51] M. Aitchison, P. Sweetser, and M. Hutter, "Atari-5: Distilling the arcade learning environment down to five games," in *International Conference on Machine Learning*. PMLR, 2023, pp. 421–438.

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

[53] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[54] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[55] J. Chen, Y. Liang, Q. Shen, J. Jiang, and S. Li, "Toward understanding deep learning framework bugs," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–31, 2023.

[56] S. Ahmed, M. Wardat, H. Bagheri, B. D. Cruz, and H. Rajan, "Characterizing bugs in python and r data analytics programs," *arXiv preprint arXiv:2306.08632*, 2023.

[57] A. Prochnow and J. Yang, "Diffwatch: watch out for the evolving differential testing in deep learning libraries," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 46–50. [Online]. Available: https://doi.org/10.1145/3510454.3516835

[58] N. Louloudakis, P. Gibson, J. Cano, and A. Rajan, "Deltann: Assessing the impact of computational environment parameters on the performance of image recognition models," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2023, pp. 414–424.

[59] S. Herbold and S. Tunkel, "Differential testing for machine learning: an analysis for classification algorithms beyond deep learning," *Empirical Software Engineering*, vol. 28, no. 2, p. 34, 2023.

[60] P. S. Kochhar, F. Thung, and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," in *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER)*. IEEE, 2015, pp. 560–564.

[61] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, 2009, pp. 57–68.

[62] 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.

[63] P. Christodoulou, "Soft actor-critic for discrete action settings," *arXiv preprint arXiv:1910.07207*, 2019.

[64] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[65] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[66] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 876–888.

[67] J. Wang, T. Lutellier, S. Qian, H. V. Pham, and L. Tan, "Eagle: creating equivalent graphs to test deep learning libraries," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 798–810.

[68] Z. Deng, G. Meng, K. Chen, T. Liu, L. Xiang, and C. Chen, "Differential testing of cross deep learning framework {APIs}: Revealing inconsistencies and vulnerabilities," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 7393–7410.

[69] C. S. Xia, M. Paltenghi, J. Le Tian, M. Pradel, and L. Zhang, "Fuzz4all: Universal fuzzing with large language models," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3597503.3639121

[70] Y. Deng, C. S. Xia, H. Peng, C. Yang, and L. Zhang, "Large language models are zero-shot fuzzers: Fuzzing deep-learning libraries via large language models," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 423–435. [Online]. Available: https://doi.org/10.1145/3597926.3598067

[71] C. Yang, Y. Deng, R. Lu, J. Yao, J. Liu, R. Jabbarvand, and L. Zhang, "White-box compiler fuzzing empowered by large language models," *arXiv preprint arXiv:2310.15991*, 2023.

[72] C. Yang, Z. Zhao, and L. Zhang, "Kernelgpt: Enhanced kernel fuzzing via large language models," *arXiv preprint arXiv:2401.00563*, 2023.

[73] H. Dai, X. Peng, X. Shi, L. He, Q. Xiong, and H. Jin, "Reveal training performance mystery between tensorflow and pytorch in the single gpu environment," *Science China Information Sciences*, vol. 65, pp. 1–17, 2022.

[74] O.-C. Novac, M. C. Chirodea, C. M. Novac, N. Bizon, M. Oproescu, O. P. Stan, and C. E. Gordan, "Analysis of the application efficiency of tensorflow and pytorch in convolutional neural network," *Sensors*, vol. 22, no. 22, p. 8872, 2022.

[75] T. Ahmad, A. Ashraf, D. Truscan, A. Domi, and I. Porres, "Using deep reinforcement learning for exploratory performance testing of software systems with multi-dimensional input spaces," *IEEE Access*, vol. 8, pp. 195 000–195 020, 2020.

[76] P. S. Nouwou Mindom, A. Nikanjam, and F. Khomh, "A comparison of reinforcement learning frameworks for software testing tasks," *Empirical Software Engineering*, vol. 28, no. 5, p. 111, 2023.

[77] J. Kim, M. Kwon, and S. Yoo, "Generating test input with deep reinforcement learning," in *Proceedings of the 11th International Workshop on Search-Based Software Testing*, 2018, pp. 51–58.