How Do Deep Learning Faults Affect AI-Enabled Cyber-Physical Systems in Operation? A Preliminary Study Based on DeepCrime Mutation Operators

Aitor Arrieta, Pablo Valle, Asier Iriarte, Miren Illarramendi
Electronics and Computer Science
Mondragon University
Mondragon, Spain
{aarrieta, millarramendi}@mondragon.edu, {pablo.valle, asier.iriarte}@alumni.mondragon.edu

***Abstract*—Cyber-Physical Systems (CPSs) combine digital cyber technologies with physical processes. As in any other software system, in the case of CPSs, the use of Artificial Intelligence (AI) techniques in general, and Deep Neural Networks (DNNs) in particular, is contantly increasing. While recent studies have considerably advanced the field of testing AI-enabled systems, it has not yet been investigated how different Deep Learning (DL) bugs affect AI-enabled CPSs in operation. This work-in-progress paper presents a preliminary evaluation on how such bugs can affect CPSs in operation by using a mobile robot as a case study system. For that, we generated DL mutants by using operators proposed by Humbatova et al., which are operators based on real-world DL faults. Our preliminary investigation suggests that such bugs are more difficult to detect when they are deployed in operation rather than when testing their DNN in an off-line setup, which contrast with related studies.**

***Index Terms*—Mutation Testing, Deep Learning, Physical Testing**

I. Introduction

Systems that combine digital cyber technologies to control physical processes are commonly known as Cyber-Physical Systems (CPSs) [1]. The autonomy of such systems is becoming higher thanks, to a large extent, to the recent advances on Artificial Intelligence (AI) techniques. Recent studies show that controllers based on Deep Neural Networks (DNNs), which are the state-of-the-art machine-learning techniques, outperform traditional CPS controllers in terms of robustness and run-time performance in several domains [2]–[4]. However, as any other software systems, AI-based techniques are not exempt of faults, which can result in irreparable losses in the context of safety and/or mission-critical CPSs.

In the last few years, testing DNN-based systems, including DNN-based CPSs, has been extensively studied. This has been tackled from different perspectives, including, the proposal of novel test adequacy criteria [5]–[11], automated test generation techniques [12]–[18], regression test optimization [19], [20], debugging and repair [21]–[26] and the test oracle problem [27]–[31]. There are also other studies that have investigated the difference between online and off-line testing of autonomous driving systems [32], [33], which are a type of AI-enabled CPSs, or the difference between testing by using simulation-based techniques and physical testing [34]. Besides, different studies have investigated the types of bugs that Deep Learning (DL) techniques suffer [35]–[38]. However, to the best of our knowledge, this is the first study that investigates how DL bugs affect AI-enabled CPSs in operation. Investigating this is paramount to prioritize research directions on testing AI-enabled CPSs.

In this work-in-progress paper, we aim at shedding light on how DL bugs affect AI-enabled CPSs in operation, and how these compare to off-line testing of the buggy DNN. In off-line testing, DNNs are tested as an independent entity operating in open-loop mode [32]. This involves providing the DNN under test with test inputs that are generated independently of the DNN, either through manual or automatic means [32]. In an off-line setup, the DNN output is commonly evaluated by comparing its predicted output to an expected test output, known as a test oracle, and the difference between the two values is the prediction error [32]. Conversely, in physical testing, DNNs are embedded in the real CPS and tested within the environment in a closed-loop mode (similar to on-line testing [32], but in a physical environment instead of a simulated one). The DNNs receive test inputs from the environment, and their outputs are fed back into the system through commands to the actuators (e.g., electrical engines). In this sense, similar to on-line testing [32], in physical testing, the DNNs are tested by monitoring the requirements that are violated (e.g., in an autonomous vehicle, going out of the road).

Such bugs may affect differently the DNN when deployed in operation as compared to when the DNN is tested in an off-line setup. This is because the decision taken by the DNN when the CPS is in operation can later be corrected in upcoming iterations, but the error can also be accumulated [32]. To understand how DL faults affect AI-enabled CPSs when these are in operation, we carry out an empirical evaluation by employing state-of-the-art DL mutation techniques. Specifically, we aim at answering the following Research Questions (RQs):

978-1-6654-5223-6/23/$31.00 ©2023 IEEE

- RQ1 — How do DL faults affect AI-enabled CPSs in operation?
- RQ2 — How do DL faults differ when deployed in an AI-enabled CPS in operation as compared to when executed in an off-line fashion?
- RQ3 — Are there differences in terms of killability between the type of DL faults when deployed in operation?

To answer these RQs, our evaluation involves a total of 20 mutants encompassing 4 different mutation operators proposed by Humbatova et al. [36] in an AI-enabled CPS involving an autonomous robot. Specifically, this autonomous robot aims at following the path of a circuit, which is delimited by two red lines. Our preliminary findings suggest that (1) detecting DL bugs is easier when being executed in an off-line mode rather than when deployed in operation, i.e., 19 out of 20 were detected off-line whereas 7 out of 20 were detected in operation; (2) certain types of DL bugs (i.e., mutation operators) are easier to detect than others.

This paper makes the following two core contributions: First, we carry out what, to the best of our knowledge, is the first empirical evaluation of how DL faults affect AI-enabled CPSs in operation. Second, we provide the source code, a new dataset, and all the trained (faulty) DNNs derived with the mutation operators proposed by Humbatova et al. [36].

The rest of the paper is structured as follows: Section II summarizes basic background on AI-enabled CPSs and how these are tested. Section III presents our experimental setup. The results of our evaluation is analyzed and discussed in Section IV. Section V summarize the main threats to validity of our study. Section VI positions our work with the current state-of-the-art. Section VII discusses next step to complete the evaluation. Lastly, we conclude our paper in Section VIII.

II. Background

In this section, we summarize basic background related to AI-enabled CPSs and how these are tested.

A. AI-enabled CPSs

Cyber-Physical Systems (CPSs) integrate digital cyber technologies (e.g., microprocessors, communication systems, software) with physical processes [1]. The autonomy of such systems is considerably increasing due to the recent advances in AI techniques. While traditional controllers (e.g., PID controllers) are widely used yet, the use of DNNs permits increasing the autonomy of such systems to control more complex tasks (e.g., autonomous vehicle functionalities). Those CPSs that employ AI techniques to control their physical processes are commonly known as AI-enabled CPSs [39]. To train DNNs, two approaches are usually employed [39]: Deep Reinforcement Learning and Supervised Learning. The former employs an agent that gets rewarded when performing well, while it gets penalized when not doing so in a "trial and error" fashion [39]. In contrast, the latter learns from historical data collected while the CPS works in operation. This study focuses on the latter one.

B. Off-line, on-line and physical testing

Off-line testing, which is also known as *"model-level testing"*, aims at testing the DNN as a standalone component based on an unseen test set [34]. In regression problems, which are commonly targeted by AI-enabled CPSs, off-line testing uses typically the mean squared error (MSE) or the mean absolute error (MAE). In contrast, for on-line testing, also known as *"system-level testing"*, the DNN is deployed and integrated with the remaining CPS. At this level, the system is tested through a set of requirements [32], [34], which often relate to safety ones.

On-line testing can either be virtual or physical. The core difference is that in the virtual the environment is simulated, whereas in the physical, the environment is real. In the virtual environment, there may even be different test levels. For instance, the Software-in-the-Loop test level simulates everything in a computer-based system. At a higher test level, i.e., the Hardware-in-the-Loop test level, the software that controls the CPS is embedded with the real target processor, whereas the environment is usually simulated with a real-time test bench.

C. Mutation testing

Mutation testing is a technique used to evaluate the adequacy of test suites [40]. It is used both, to guide the generation of test cases and support experimentation. It follows the intuition of seeding an (artificial) fault into the original program. A test is said to detect (or kill) that fault if the outcome of the original and the seeded programs differ. Despite this technique having been applied for years [41], its notion has been proposed to be changed for the context of DNNs [9], [10]. We give further details in the upcoming sections.

III. Evaluation Setup

We now explain the experimental setup carried out to answer the three research questions introduction in the first section of the paper.

A. Case Study System

Our case study system involves an autonomous rover, named LeoRover, which perceives as input an image of the environment that is fed to a DNN model. Figure 1 shows an image of our rover. As an outcome, the DNN model returns two values: (1) the linear velocity and (2) the angular velocity that the robot should have. These values are translated as commands, which are sent to the rover's engine controllers through a Robotic Operating Systems (ROS)-based communication system.

Although the LeoRover team provides a dataset and a set of pre-trained models [42], we obtained a new dataset. Specifically, we found that the original dataset was not robust enough to withstand slightly complex manoeuvres. To obtain this dataset, we connected the remote controller of a PlayStation to the robot, which controlled the rover's linear and angular speed values. Meanwhile, the images were stored together with such commands in a RaspberryPi the rover has on-board. In total,

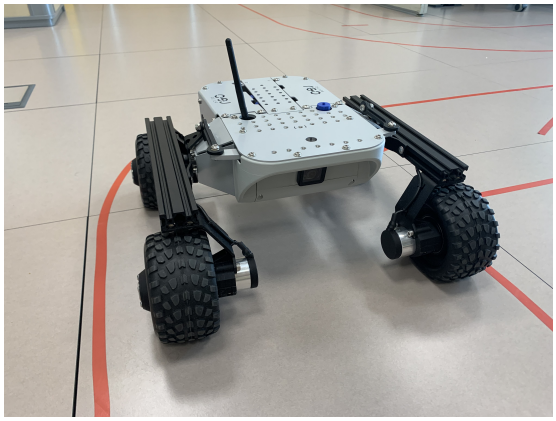

Fig. 1. The autonomous mobile robot, named LeoRover, used as a case study system in our evaluation

we obtained 16,048 images, divided into training (80% of the images) and testing dataset (20% of the images). For the training dataset, 75% were employed for the training and 25% for the validation.

B. Generation of Mutants

We generated a total of 20 mutants encompassing 4 mutation operators targeting two groups: (1) training data and (2) hyper parameters. Each of these four mutation operators were configured with five independent configurations. Below we explain the four mutation operators, which were selected because in the original work [10] they were not in the group of non killable mutants, they did not have a high triviality score and they did not produce redundant mutants:

- New Learning Rates (HLR): This mutation operator aims at changing the learning rate of the algorithm. The original learning rate was 0.001. We generated five different mutants by changing the learning rate (LR) to the following values: 0.0001, 0.0002, 0.0005, 0.005 and 0.01.
- New Number of Epochs (HNE): This mutation operator aims at changing the number of epochs for which a model is trained. The original setup involved a total of 100 epochs, although the training process did not exceed 40 epochs as the stopping criteria were met. Our mutants employed the following number of epochs (NoE): 2, 5, 8, 12 and 15 epochs.
- Add noise to training data (TAN): This operator introduces low quality training data by adding some noise to the original data. Specifically, we mutate some pixels of the input images to the neural network. Our mutants were generated by randomly mutating selected pixels. We mutated the following percentage of pixels (PoP) from an image: 5, 10, 15, 20 and 25%. It is expected that, the higher this percentage, the easier the mutant is detected.
- Change labels of training data (TCL): This operator mutates the labels of the data, mimicking situations of wrong labels provided to the training algorithms. We randomly change the linear and angular speeds of some of the labels in the training dataset. The linear velocity was assigned a random number between 0 and 0.5, whereas the angular velocity a random number between -2 to 2. We mutate 15, 20, 25, 30 and 35% of images (PoI) with new values.

Because the generation of these mutants involves new training, and such training is stochastic, the generation of these mutants requires generating multiple models [10]. Therefore, each mutant was generated 10 times, encompassing in total 200 trained models. Note that we could not afford more generations because the execution with the physical autonomous robot required significant manual effort.

Moreover, the notion of mutation killing must be defined using a statistical comparison between the distribution of the output of the original neural network and the output distribution of the mutant, as originally proposed by Jahangirova and Tonella [9]:

$$isKilled(P, M, TestD) = \begin{cases} true & \text{if effect size} \geq \beta \\ & \text{and p\_value} < \alpha \\ false & \text{otherwise} \end{cases} \quad (1)$$

where P stands for $P_1, P_2, ..., P_n$ original neural network models and M for $M_1, M_2, ..., M_n$ mutated models [9]. Meanwhile, as in [9], each couple (P_i, M_i) is evaluated using test data $TestD_i$, $i \in [1 : n]$ (see Section III-C). As previously mentioned, in our study, we used $n = 10$. Similar to the paper by Humbatova et al. [10], we used the generalised linear model (GLM) for the calculation of the statistical significance (with $\alpha = 0.05$) and the Cohen's d (with $\beta = 0.5$) for the effect size.

C. Evaluation Metrics

On the one hand, for off-line testing, we used the Mean Squared Error (MSE) values of the two variables that the DNN inferred over a set of images from the testing dataset. On the other hand, for the physical validation, we first built a circuit. To build such circuit, the following criteria were necessary: (1) the robot would need to have forward, leftward and rightward movements; and (2) the light of the environment in which the robot is executed had to be constant. This way, all mutated and non-mutated models are executed under the same environmental conditions. Based on this, we built the circuit depicted in Figure 2. The goal of the rover was to give two entire laps. As evaluation metric for the angular values provided by the DNN, we visually controlled that the rover was not deviating enough (i.e., all the four wheels were out of the path). For the linear velocity values, we measured the time that the rover along with the deployed DNN models took to complete two entire laps. It is noteworthy that if the robot deviates too much, going out of the path, the entire laps are uncompleted. In such a case, we marked the time as 0 seconds, which is sufficiently distinct. As the time was measured manually, two of the authors recorded the time and the average of both was taken.

TABLE I
SUMMARY OF THE RESULTS FOR EACH MUTANT (10 DIFFERENT REPETITIONS) WHEN DEPLOYED IN OPERATION (PHYSICAL TESTING) OR IN OFF-LINE.

		Physical Testing							**Off-line Testing**						
		Linear Speed			**Angular Speed**			**Mutant**	**Linear Speed**			**Angular Speed**			**Mutant**
	Configuration	**Killed**	**p-val**	**d-cohen**	**Killed**	**p-val**	**d-cohen**	**Killed**	**Killed**	**p-val**	**d-cohen**	**Killed**	**p-val**	**d-cohen**	**Killed**
HLR	LR = 0.0001	False	0.2544	-0.5265	False	0.2878	-0.4899	False	True	0.001	1.727	True	0.0003	1.9820	True
	LR = 0.0002	False	0.2660	-0.5134	False	0.2878	-0.4899	False	False	0.062	0.889	True	0.0053	1.4175	True
	LR = 0.0005	False	0.5187	-0.2944	False	0.5560	-0.2683	False	False	0.510	0.301	False	0.9411	0.0335	False
	LR = 0.005	True	<0.0001	-2.4083	True	<0.0001	-2.5298	True	True	0.008	1.330	True	0.0316	1.0426	True
	LR = 0.01	True	<0.0001	-4.0018	True	<0.0001	-4.0249	True	True	<0.0001	44.0822	True	<0.0001	38.8273	True
TAN	PoP = 5%	False	0.5220	-0.2920	False	0.5560	-0.2683	False	True	0.010	1.2977	True	0.0006	-1.8504	True
	PoP = 0.1%	False	0.9467	-0.0303	False	1	0.0000	False	True	0.001	1.7995	False	0.4304	-0.3607	True
	PoP = 0.15%	False	0.9561	0.0249	False	1	0.0000	False	True	<0.0001	2.2397	True	0.0033	1.5150	True
	PoP = 0.2%	False	0.8409	-0.0911	False	0.5560	-0.2683	False	True	<0.0001	2.9245	True	<0.0001	5.0658	True
	PoP = 25%	False	0.2593	-0.5209	False	0.1346	-0.7006	False	True	<0.0001	4.7084	True	<0.0001	7.6013	True
TCL	PoI = 15%	True	<0.0001	-2.5252	True	<0.0001	-2.5298	True	True	<0.0001	11.7770	True	<0.0001	16.4045	True
	PoI = 2%	True	<0.0001	-4.0018	True	<0.0001	-4.0249	True	True	<0.0001	13.9346	True	<0.0001	15.5744	True
	PoI = 25%	True	<0.0001	-4.0018	True	<0.0001	-4.0249	True	True	<0.0001	19.8078	True	<0.0001	13.8662	True
	PoI = 3%	True	<0.0001	-4.0018	True	<0.0001	-4.0249	True	True	<0.0001	12.2734	True	<0.0001	18.3378	True
	PoI = 35%	True	<0.0001	-4.0018	True	<0.0001	-4.0249	True	True	<0.0001	15.0016	True	<0.0001	16.9399	True
HNE	NoE = 2	False	0.3208	-0.4566	False	0.1346	-0.7006	False	True	<0.0001	4.8816	True	<0.0001	6.4894	True
	NoE = 5	False	0.2680	-0.5112	False	0.2878	-0.4899	False	True	<0.0001	4.4124	True	<0.0001	3.8986	True
	NoE = 8	False	0.2419	-0.5412	False	0.2878	-0.4899	False	True	<0.0001	3.0027	True	<0.0001	2.3360	True
	NoE = 12	False	0.2478	-0.5342	False	0.2878	-0.4899	False	True	0.0004	1.9242	False	0.0808	0.8275	True
	NoE = 15	False	0.0563	-0.9124	False	0.0544	-0.9204	False	True	0.0190	1.1526	False	0.1359	0.6982	True

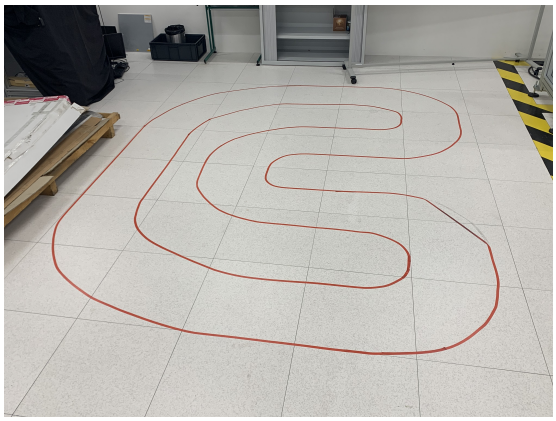

Fig. 2. Circuit used in our experiments

IV. ANALYSIS OF THE RESULTS AND DISCUSSION

We now analyze the results of the carried out evaluation and discuss them.

A. RQ1 — Faults in operation

Within the first RQ, we aimed at evaluating how different faults affects an AI-enabled CPS in operation. We found that 7 out of 20 mutants (35%) affected the rover sufficiently in order this not to successfully complete the laps. Indeed, in all of these cases, the rover went out of bounds. Conversely, for those laps that were completed (i.e., the rover did not go out of bounds), the time required for the rover to complete the laps did not suffer significant changes that would enable killing any mutant. A possible explanation for this could be that the problem for inferring the angular velocity is more complex than the problem for inferring the linear velocity. Therefore, we can conclude that the majority of the mutants were not detected when deployed in the physical rover, answering the first RQ as follows:

RQ1: 35% of the mutants were detected as these violated the requirement of following the lines and therefore going out of bounds. For the remaining cases, the mutants did not affect the behavior of the CPS.

B. RQ2 -– Difference between faults in operation or when off-line testing

The second RQ aimed at assessing the difference between executing the mutants in an off-line fashion (i.e., the DNN isolated from the CPS) or in operation. We found that, when executing the mutants off-line, as carried out in previous works [36], the detection rate significantly increased. In fact, 19 out of 20 mutants (95%) were killed. Interestingly, more mutants (18 out of 20) were killed by means of the linear speed inference variable than by means of the angular speed inference (16 out of 20 mutants). These results contrast with the results obtained when the DNN is deployed in operation, where there was no mutant detected based on the metric of measuring the time to complete two laps. Furthermore, our results also contrast with the findings from a prior work, in which off-line testing is compared with on-line [32]. In that study, the authors suggested that *"Offline testing is more optimistic than online testing because the accumulation of errors is not observed in offline testing"* [32].

In conclusion, based on our preliminary findings, detecting faults in an off-line setup through the test dataset seems to be easier than detecting them in operation. A potential reason could be that a sub-optimal decision in operation could be eventually corrected within the following iterations of the execution of the DNN, which is executed in a closed-loop setup. Conversely, when testing the DNN model off-line, the fault is accumulated when doing the inference, which results in higher MSE values. We can therefore answer the second RQ as follows:

RQ2: The DNN mutants are easier to be killed in an off-line setup through the test dataset than when deployed in operation in the AI-enabled CPS. 95% of the mutants were killed when testing the DNN mutants off-line as compared when testing them by deploying them in the real CPS.

C. RQ3 – Difference between DL fault types

The third RQ aimed at assessing how different DL fault types behaved when deployed in a AI-enabled CPS. Two of the four types of analysed mutation operators were not detected for any of the configurations: TAN and HNE. HLR was detected for 2 out of the 5 selected hyperparameters (i.e., when the learning rate was 0.05 and 0.1). Lastly, TCL was detected for all the selected hyperparameter configurations, which suggests that mutating the labeling of the data with wrong values has a large impact on the DNN. Therefore, one core conclusion of this study is that the data needs to be carefully cleaned before training the model. In summary, we can conclude that different faults behave differently. Thus, the third RQ can be answered as follows:

RQ3: Different types of faults behave differently. Among the four analysed mutation operators, HNE was the most easily killable one when deploying the mutants in the physical rover, followed by the HLR. Conversely, the TAN and HNE mutation operators were not detected when testing the rover in the physical system.

V. Threats to Validity

We now summarize the threats of our study and how we tried to mitigate them.

Internal: As the employed case study system did not have a dataset of real faults available, an *internal validity threat* of our work could be related to the injected faults, which are not real. Instead, we relied on the mutation operators proposed by Humbatova et al. [10], which are based on real faults.

External: An *external validity threat* of our evaluation could be related to the case study system we used, which may not generalize to all AI-enabled CPS. Instead, we focused on a case study system involving a lane-keeping DNN, which is similar to other related work employing AI-enabled CPSs [33]. While we believe that further case studies are needed to validate our conclusions, we believe that our findings may be applicable to other AI-enabled CPSs whose control systems involve a closed-loop behavior.

Conclusion: A *conclusion validity threat* of our study relates to the non-deterministic nature of the DL training process. This threat was mitigated by generating, for each mutant, 10 different models and applying the notion of mutation killing based on statistical significance introduced by previous works [9], [10]. In addition, for the metrics employed during physical testing, the time that took the rover to complete two complete laps was manually recorded. To mitigate bias in the recording times, two of the authors recorded the laps independently and the average of both was taken into account.

Construct: A *construct validity threat* of our study could relate to the environmental effects (e.g., light, weather conditions) that could affect the outcome of the DNN that was embedded in the physical rover. We mitigated this threat by building the circuit into a closed room without windows and maintaining a constant light for all mutants and models.

VI. Related Work

Some recent papers have studied the difference between testing AI-enabled CPSs off-line, on-line, in simulated and in physical environments. This was first pioneered by Haq et al., [32], who compared the differences between off-line testing with on-line (simulation-based) testing of DNNs in the field of autonomous cars. The same authors later extended this study by including new DNNs, added additional analysis and extended their empirical findings [33]. Their findings contrast with ours in the sense that they claim that off-line testing is more optimistic than on-line. Besides these claims, our evaluation differs from different perspectives with respect to both papers by Haq et al., [32], [33]. Firstly, they employ pre-trained DNNs, whereas in our case, we use faulty DNNs that are trained by using realistic mutation operators [10]. Secondly, their on-line setup is carried out in a simulated environment, whereas ours is in a real physical environment. These studies were later replicated by Stocco et al. [43] with the goal of mitigating three identified core threats and employing a physical self-driving car during the evaluation. In contrast to our study, Stocco et al. [43] do not study the influence of DL faults.

In another study, Stocco et al., investigated the transferability between simulation-based testing to real-world testing [34]. Their findings suggest that some DNN failures can be exposed only in real-world conditions. Unlike our study, which focuses on faults on the DL system, Stocco et al. [34] inject faults by corrupting images (e.g., including noise, saturating the image or including fog). Unlike our study, which requires training different DNN models for injecting faults, their DNNs are pre-trained state-of-the-art models.

A recent study investigated [44] testing of AI-enabled CPSs from different perspectives. Specifically, the authors (1) compared how deep reinforcement learning-based controllers performed with respect to traditional controllers; (2) whether existing CPS testing methods are still effective on AI-based CPSs; and (3) whether the combination of traditional and AI controllers can bring better performance to CPSs. Our study differs significantly from this one in different aspects. For instance, we focus on supervised learning approaches instead of deep reinforcement learning. Also, our goal was not to compare different testing methods but how deep learning faults affects AI-enabled CPSs in operation.

Several studies have investigated the type of bugs in the domain of machine learning from different perspectives. One of such seminal studies is that of Thung et al., [35], which investigated machine-learning bugs by analyzing three systems

(i.e., Apache Mahout, Lucene and OpenNLP); their bugs were classified in different categories (e.g., algorithmic faults, non-functional faults). Humbatova et al. [36] built a large taxonomy of deep learning systems faults by manually analyzing 1,059 artefacts, posts from Stack Overflow and interviewing 20 researchers and practitioners. Sun et al., [37] conducted a study of machine-learning bugs on Github programs by analyzing 329 closed bugs, in which they studied their category, fix pattern, fix scale, fix duration and the type of software maintenance. Islam et al., [38] studied a total of 2,716 posts from Stack Overflow and 500 commits that fix bugs of five popular deep learning libraries, concluding that two types of bugs (i.e., data bug and logic bug) were the most severe bug types in deep learning software applications. Ninkanjam et al., [45] analyzed the type of faults that appeared in deep reinforcement learning systems. All these studies focused on what type of faults are exhibited in AI-based software. Conversely, our study focuses on how DL bugs affect AI-enabled CPSs when these are deployed in operation.

VII. Next Steps and Future Work

In this paper, we communicate our initial research results of how DL faults affect AI-enabled CPSs. Still, further work and steps are necessary towards a more complete evaluation. We now discuss key steps to be taken in the future:

Other faults: In this study, we limited our empirical evaluation to 20 mutants encompassing 4 different fault types instantiated each to 5 configurations. Other type of mutation operators are required to further confirm the claims we do in this study.

Other case study systems: We used a mobile robot as a case study system, but it is difficult to generalize the findings to any type of CPS with a single case study. In the future, we would like to contrast the results of this paper with other type of CPS (e.g., an unmanned aerial vehicle, a robotic arm, an inverted pendulum). It is noteworthy, however, that experimenting with such type of case study systems is expensive, not only because of the cost of the system itself, but also because significant manual labor is required to execute the physical tests, as this is not possible to automate.

Other control levels: In the context of CPSs, the control systems can be abstracted into several levels. For instance, low-level controllers take low-level decisions (e.g., in our mobile robot case study, controlling the speed of the engine of the rover), whereas higher-level controllers are more concerned towards strategic decisions (e.g., in an autonomous vehicle, which route should be the best one). AI techniques can be embedded at different control levels of an AI-enabled CPS. It could be interesting to study whether different bugs at different levels affects differently to the system.

Difference with a simulated environment: In this study, we have compared how off-line testing differs with physical testing, which is a type of on-line testing method. However, it could be interesting to assess whether DL faults in operation can be detected in (on-line) simulation and vice-versa, as the texture of the images of a simulated environment is different from those that appear in a real physical system. This could have significant implications from both, research as well as practical perspectives. From the research perspective, many of the research carried out is subject to the assumption that the employed simulators are realistic enough to mimic real-world scenarios. From the practical perspective, automating the test process by employing simulation-based testing is easier and less expensive, therefore, confirming that DL faults can be detected through a simulator would have positive implication in the CPS industry.

VIII. Conclusion

This paper presents a preliminary study on how DL faults affect AI-enabled CPSs when deployed in operation. To this end, we made use of realistic mutation operators proposed by Humbatova et al., [10]. Our preliminary findings suggest that detecting DL faults off-line, through the testing dataset, is easier than detecting them in operation. If such a finding is further confirmed through a stronger experimental set-up (e.g., with more faults, more AI-enabled CPSs), this could have important implications in the research community, which could focus their effort on leveraging techniques for testing DNN in an off-line fashion. However, there are other important underlying implications. For instance, the dataset for off-line testing might be poor, or may not consider unforeseen scenarios. This would imply further physical testing, which is extremely expensive.

Replication Package

The replication package of our study can be found in the following link: https://doi.org/10.5281/zenodo.8126181

Acknowledgments

This work was partially founded by the Basque Government through their Elkartek program (EGIA project, ref. KK-2022/00119 and SIIRSE project, ref. KK-2022/00007). The authors are part of the Software and Systems Engineering research group of Mondragon Unibertsitatea (IT1519-22), supported by the Department of Education, Universities and Research of the Basque Country.

References

[1] P. Derler, E. A. Lee, and A. S. Vincentelli, "Modeling cyber–physical systems," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 13–28, 2011.

[2] S. Alsalehi, N. Mehdipour, E. Bartocci, and C. Belta, "Neural network-based control for multi-agent systems from spatio-temporal specifications," in *2021 60th IEEE Conference on Decision and Control (CDC)*. IEEE, 2021, pp. 5110–5115.

[3] W. Liu, N. Mehdipour, and C. Belta, "Recurrent neural network controllers for signal temporal logic specifications subject to safety constraints," *IEEE Control Systems Letters*, vol. 6, pp. 91–96, 2021.

[4] S. Yaghoubi, G. Fainekos, and S. Sankaranarayanan, "Training neural network controllers using control barrier functions in the presence of disturbances," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–6.

[5] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 2018, pp. 120–131.

[6] Z. Yang, J. Shi, M. H. Asyrofi, and D. Lo, "Revisiting neuron coverage metrics and quality of deep neural networks," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 408–419.

[7] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?" in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 851–862.

[8] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*. IEEE, 2018, pp. 100–111.

[9] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 74–84.

[10] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 67–78.

[11] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1039–1049.

[12] L. Ma, F. Juefei-Xu, M. Xue, B. Li, L. Li, Y. Liu, and J. Zhao, "Deepct: Tomographic combinatorial testing for deep learning systems," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 614–618.

[13] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Deephyperion: exploring the feature space of deep learning-based systems through illumination search," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 79–90.

[14] 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.

[15] ——, "When and why test generators for deep learning produce invalid inputs: an empirical study," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.

[16] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: a coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.

[17] F. U. Haq, D. Shin, and L. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.

[18] ——, "Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 811–822.

[19] M. Weiss and P. Tonella, "Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 139–150.

[20] A. Arrieta, "Multi-objective metamorphic follow-up test case selection for deep learning systems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 1327–1335.

[21] M. Sotoudeh and A. V. Thakur, "Provable repair of deep neural networks," in *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2021, pp. 588–603.

[22] M. J. Islam, R. Pan, G. Nguyen, and H. Rajan, "Repairing deep neural networks: Fix patterns and challenges," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1135–1146.

[23] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search based repair of deep neural networks," *ACM Trans. Softw. Eng. Methodol.*, sep 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3563210

[24] M. Wardat, B. D. Cruz, W. Le, and H. Rajan, "Deepdiagnosis: Automatically diagnosing faults and recommending actionable fixes in deep learning programs," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 561–572.

[25] Y. Li, M. Chen, and Q. Xu, "Hybridrepair: towards annotation-efficient repair for deep learning models," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2022, pp. 227–238.

[26] S. Tokui, S. Tokumoto, A. Yoshii, F. Ishikawa, T. Nakagawa, K. Munakata, and S. Kikuchi, "Neurecover: Regression-controlled repair of deep neural networks with training history," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 1111–1121.

[27] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[28] A. Dwarakanath, M. Ahuja, S. Sikand, R. M. Rao, R. J. C. Bose, N. Dubash, and S. Podder, "Identifying implementation bugs in machine learning based image classifiers using metamorphic testing," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 118–128.

[29] J. Ding, X. Kang, and X.-H. Hu, "Validating a deep learning framework by metamorphic testing," in *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*. IEEE, 2017, pp. 28–34.

[30] Z. Zhang, P. Wang, H. Guo, Z. Wang, Y. Zhou, and Z. Huang, "Deepbackground: Metamorphic testing for deep-learning-driven image recognition systems accompanied by background-relevance," *Information and Software Technology*, vol. 140, p. 106701, 2021.

[31] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 359–371.

[32] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Comparing offline and online testing of deep neural networks: An autonomous car case study," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 85–95.

[33] F. U. Haq, D. Shin, S. Nejati, and L. Briand, "Can offline testing of deep neural networks replace their online testing? a case study of automated driving systems," *Empirical Software Engineering*, vol. 26, no. 5, p. 90, 2021.

[34] A. Stocco, B. Pulfer, and P. Tonella, "Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2022.

[35] F. Thung, S. Wang, D. Lo, and L. Jiang, "An empirical study of bugs in machine learning systems," in *2012 IEEE 23rd International Symposium on Software Reliability Engineering*. IEEE, 2012, pp. 271–280.

[36] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1110–1121.

[37] X. Sun, T. Zhou, G. Li, J. Hu, H. Yang, and B. Li, "An empirical study on real bugs for machine learning programs," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 348–357.

[38] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 510–520.

[39] D. Lyu, J. Song, Z. Zhang, Z. Wang, T. Zhang, L. Ma, and J. Zhao, "Autorepair: Automated repair for ai-enabled cyber-physical systems under safety-critical conditions," *arXiv preprint arXiv:2304.05617*, 2023.

[40] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: an analysis and survey," in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 275–378.

[41] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[42] L. Rover, "Leorover," https://github.com/LeoRover, accessed on May 2, 2023.

[43] A. Stocco, B. Pulfer, and P. Tonella, "Model vs system level testing of autonomous driving systems: A replication and extension study."

[44] J. Song, D. Lyu, Z. Zhang, Z. Wang, T. Zhang, and L. Ma, "When cyber-physical systems meet ai: a benchmark, an evaluation, and a way forward," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, 2022, pp. 343–352.

[45] A. Nikanjam, M. M. Morovati, F. Khomh, and H. Ben Braiek, "Faults in deep reinforcement learning programs: a taxonomy and a detection approach," *Automated Software Engineering*, vol. 29, no. 1, p. 8, 2022.