Difficulty and Severity-Oriented Metrics for Test Prioritization in Deep Learning Systems

Hamzah Al-Qadasi, Yliès Falcone, Saddek Bensalem

Difficulty and Severity-Oriented Metrics for Test Prioritization in Deep Learning Systems

Hamzah Al-Qadasi[1], Yliès Falcone[2], Saddek Bensalem[1]
[1]*Univ. Grenoble Alpes, CNRS, Grenoble INP, Verimag, 38000 Grenoble, France*
[2]*Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, 38000 Grenoble, France*

{hamzah.al-qadasi, ylies.falcone, saddek.bensalem}@univ-grenoble-alpes.fr

0 ***Abstract*—Recently, there has been a growing trend in AI testing toward developing new test prioritization algorithms for deep learning systems. These algorithms aim to reduce the cost and time needed to annotate test datasets by prioritizing instances with a higher chance of exposing faults. Various metrics have been used to evaluate the effectiveness of these algorithms, e.g., APFD, RAUC, and ATRC. However, there is a lack of research to confirm their validity. The results indicate that the existing metrics have severe limitations. For example, some metrics ignore the labeling budget and prioritize the fault detection rate instead of the fault detection ratio. Moreover, others overlook the prioritization difficulty in the evaluation.**

As a solution, we develop a new metric (WFDR), which solves the deficiencies of previous metrics. We also draw attention to a new research area, known as *severity prioritization*, which emphasizes the importance of prioritizing misclassified instances according to the severity level, particularly in critical situations. Our experiments reveal that instances with high severity make up more than 20% of all misclassified instances. Thus, these instances should be prioritized when it comes to labeling. Consequently, we proposed a new metric known as (SFDR) that evaluates the effectiveness of algorithms in prioritizing high-severity instances. Our evaluations show that our proposed metrics are more effective than other existing metrics. Besides, our two metrics re-evaluate some recent algorithms and indicate that these algorithms perform poorly.

***Index Terms*—Test Prioritization Algorithms, Evaluation Metrics, Severity Prioritization, Effective Labeling.**

I. Introduction

We are witnessing an unprecedented increase in the volume of big data. Nevertheless, this swift growth presents a significant challenge: the demand for data labeling. Manual labeling can be excessively costly in terms of both time and resources. The labeling process can be more challenging in multi-label classification, object detection, and segmentation.

Test prioritization is a research area where algorithms alleviate the issue of manual labeling in big data. Prioritization algorithms also help deep learning specialists to evaluate the performance of the trained model on a carefully selected test dataset rather than a randomly selected one. More specifically, these techniques mainly focus on prioritizing error-revealing examples which expose the vulnerabilities of the trained neural network. As a result, these techniques reduce the labeling cost and also make the testing process more effective and efficient.

A plethora of test prioritization algorithms have been recently published, e.g., DeepGini [1],[2], Neurons Pattern [3], TestRank [4], TPFL [5], PRIMA [6], ActGraph [7], Predictive Mutation [8], and DeepAbstraction [9]. These algorithms are evaluated by the following metrics: APFD [10], RAUC [11], and ATRC [4]. These metrics are widely adopted, however, they suffer from critical limitations, leading to misleading conclusions.

For instance, the APFD metric is widely used to evaluate the prioritization algorithms in software testing. More importantly, there are significant differences between deep learning and software systems, e.g., the bug nature and the system structure. Since there is no labeling for test cases in software testing, the APFD metric neglects the cost of labeling in its formula. This critical oversight leads to consistently overestimating the algorithm's effectiveness in deep learning systems.

In addition, RAUC and ATRC metrics mainly evaluate the algorithms on the fault detection rate, not the fault detection ratio. In other words, algorithms prioritizing quickly a few error-revealing instances are more effective than the ones prioritizing numerous error-revealing instances slowly. These metrics also neglect the prioritization difficulty, which varies during the prioritization.

Thus, we propose a metric called WFDR to solve the drawbacks of the existing metrics. The WFDR metric evaluates the algorithms according to both criteria: the fault detection ratio and rate. Also, WFDR is a weighted metric, which involves the prioritization difficulty formally represented by the adaptive weights.

In critical safety and security scenarios, algorithms should prioritize misclassified instances with high prediction probability over all misclassified ones. But the proceeding metrics handle all prioritized error-revealing instances similarly, despite the various levels of severity. Hence, we present a novel metric (SFDR) that evaluates how effectively the algorithms prioritize high-severity instances. In the real world, these instances can lead to devastating outcomes since we often rely on classifications with high certainty.

This paper is supported by the European Horizon 2020 research and innovation programme under grant agreement No. 956123 and by the French National Research Agency (ANR) in the framework of the Investissements d'Avenir program (ANR-10-AIRT-05, irtnanoelec).

In this paper, we summarize our significant contributions as follows:

- We conduct the first intensive study to investigate the effectiveness of the existing metrics.
- We develop a novel metric (WFDR) that solves the limitations of the predecessors.
- We develop a new metric (SFDR) that evaluates algorithms in the context of severity prioritization.

We structure the paper as follows: Sec. II describes the limitations of current metrics. Sec. III introduces the new metrics with equations and examples, and Sec. IV details the experiments conducted to evaluate the metrics and algorithms. The results of these experiments are discussed in Sec. V, and Sec. VI summarizes the findings and suggests potential next steps.

The interchangeable use of the terms: misclassified instances, error-revealing examples, faults, and bugs.

II. Preliminaries

This section introduces an essential background to understand the existing metrics and explains the drawbacks of each metric through some scenarios.

A. APFD

The **A**verage **P**ercentage of **F**aults **D**etection (APFD) is a metric that primarily evaluates the performance of prioritization algorithms in the software testing domain [10]. We compute the APFD value by the following equation:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF_i}{mn} + \frac{1}{2n} \tag{1}$$

where n is the total number of test cases, and m is the number of faults in the test dataset. TF_i is the order of a test case that exposes the fault(i).

The APFD value ranges between 0 and 1. When the ATPF value is zero, the algorithm works ineffectively, i.e., the order of the test cases which expose faults (TP_i) at the end of the test dataset and vice versa. For instance, if the number of faults is 10 in the 10000-test dataset and the order of the test cases that expose the faults is the last ten between 9990 and 10000. The ATPF is almost zero, which indicates the slow faults detection rate.

Since there is no labeling for test cases in software testing, the APFD metric completely ignores the labeling budget. The following example shows how ineffectively APFD evaluates the algorithm with a predefined budget.

Example 1: Unlabeled test dataset has 1000 test cases in which 100 error-revealing test cases are prioritized between 101 and 200. The APFD value is calculated as follows:

$$APFD_1 = 1 - \frac{101+...+200}{100*1000} + \frac{1}{2*1000} = 0.85$$

According to the labeling budget (100), we should label the first 100 test cases which expose faults. Thus, the correct value of APFD should be *zero*. Therefore, the APFD over-evaluates erroneously the performance of the prioritization algorithm from 0.0 to 0.85.

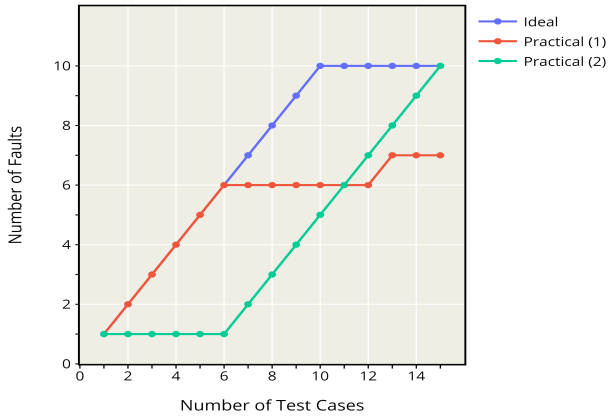

Fig. 1: RAUC metric evaluates 2 test prioritization algorithms: Practical (1) & Practical (2)

B. RAUC

The **R**atio **A**rea **U**nder the **C**urve (RAUC) is the ratio between the area under the actual performance curve to the area under the ideal curve. We compute the RAUC value by the following formula:

$$RAUC = \frac{\sum_{i=1}^{m} Practical_i}{\sum_{i=1}^{m} Ideal_i} * 100\% \tag{2}$$

where m is the labeling budget. The RAUC ratio is between 0% and 100%, where 0% indicates that all error-revealing test cases are not prioritized within the budget and vice versa.

The main issue with the RAUC metric is that it evaluates the prioritization algorithm on how quickly an algorithm prioritizes the faults, namely, **Faults Detection Rate (FDRE)**. However, the faults detection rate should not be the only contributing factor in evaluating the performance. But the metric should also involve how many erroneous test cases the algorithm prioritizes, namely, **Faults Detection Ratio (FDRO)**. The following example shows the importance of the latter factor.

Example 2: A 100-unlabeled test dataset has ten error-revealing test cases. The labeling budget is 15 test cases. We have two different algorithms: Practical-1 and Practical-2. The former algorithm has a high FDRE, and the latter has a high FDRO. The RAUCS for both algorithms are as follows:

$$RAUC_1 = \frac{78}{105} = 74.29\%, \qquad RAUC_2 = \frac{60}{105} = 57.14\%$$

Figure 1 demonstrates that the first algorithm prioritizes the first six test cases that expose faults quickly. On the other hand, the second algorithm starts poorly prioritizing test cases. But after the 6th test case, all test cases expose the faults. Hence, the latter algorithm has a higher FDRO than the former. However, the RAUC metric incorrectly evaluates the first algorithm as more effective, with a substantial margin reaching up to 17.15%. Therefore, the evaluation of RAUC is misleading and should be corrected.

C. ATRC

The **A**verage **T**est **R**elative **C**overage (ATRC) is formulated by TestRank[4]. The ATRC metric involves the labeling budget in the calculation. We compute the ATRC value by the following formula:

$$TRC = \frac{\text{\#Detected Faults}}{\min(\text{\#Labeling Budget, \#Total Faults})}$$
$$ATRC = \frac{1}{m}\sum_{i=1}^{m} TRC_i * 100\% \quad (3)$$

where TRC is the ratio between the number of detected faults to the minimum of the labeling budget and the total number of faults in the test dataset. The ATRC is the average of TRC when the labeling budget (m) is less than or equal to the total number of faults in the dataset. The ATRC metric is more effective than the APFD metric since the ATRC evaluates the performance of an algorithm under a limited budget rather than the entire dataset. In other words, the ATRC metric is a stress test within a limited budget. The following example demonstrates how the ATRC metric behaves over two datasets: (i) Dataset A has a high FDRE, and (ii) Dataset B has a high FDRO.

Example 3: A 100-unlabeled test dataset has ten error-revealing test cases, and the labeling budget is 30. Moreover, two algorithms prioritize the test dataset, which results in two prioritized datasets, namely A and B. Dataset A is [1, 1, 1, 1, 1, 1, 0, 0, 0, 0], and dataset B is [1, 0, 0, 1, 1, 1, 1, 1, 1, 1] where one represents faults, and zero represents non-faults. We compute the ATRC for both datasets as follows:

$$ATRC_1 = \frac{1}{10} * \left[\frac{1}{1} + \frac{2}{2} + ... + \frac{6}{7} + \frac{6}{8} + \frac{6}{9} + \frac{6}{10}\right] = 88.74\%$$
$$ATRC_2 = \frac{1}{10} * \left[\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{2}{4} + \frac{3}{5} + ... + \frac{8}{10}\right] = 66.42\%$$

We should calculate the ATRC value when the labeling budget is less than or equal to the number of faults (10). We can see that the ATRC metric suffers from the same problem as the RAUC metric. More specifically, the ATRC metric evaluates algorithms only on FDRE, not FDRO. Thus, the *ATRC* metric evaluates the first and second algorithms with 88.74%, and 66.42%, respectively. We can also observe clearly how the ATRC metric falsely over-evaluates the first algorithm over the second algorithm, with a significant difference up to 22.32%. Thus, the current metrics are inadequate, and new metrics should be developed.

III. Approach

In this section, we introduce the idea of the misclassification ratio with multiple examples. Then, we propose our two new metrics with the corresponding properties.

A. Misclassification ratio

The *misclassification ratio* is the ratio between the number of misclassified instances and the size of the test dataset, ranging from 0 to 1. A low ratio indicates that there are relatively few misclassified instances compared to the overall size of the dataset, making it difficult for the algorithm to prioritize these few instances and vice versa. As a result, the difficulty of prioritization is inversely proportional to the misclassification ratio. As such, it is unfair to evaluate the algorithm's prioritization capability without considering the misclassification ratio. For instance, a 100-test dataset has 20 misclassified instances. The initial misclassification ratio is 20/100. If the first prioritized instance is an error-revealing instance, the misclassification ratio becomes 19/99. If the first 19 instances are misclassified, it is challenging to prioritize the last misclassified instance among the 81-test dataset.

B. Motivational Example

Example 4: Datasets A and B have 14 and 1000 test instances, respectively. Each dataset has eight misclassified instances. A particular algorithm prioritizes both datasets as follows: [1,1,1,1,0,0,0,0]. We find that $RAUC_A$ and $RAUC_B$ are 77.22%, and $ATRC_A$ and $ATRC_B$ are 81.73%.

However, datasets A and B have different misclassification ratios, each metric evaluates the performance equally in both datasets. Typically, the algorithm performance in dataset B should be higher than in dataset A since the prioritization process is more difficult. As a result, we need to develop an evaluation metric that considers the misclassification ratio.

C. Weighted Faults Detection Ratio

The process of prioritization typically becomes more challenging as it progresses. Thus, it is necessary to assign weights at each step of the process. These weights should gradually increase with each successful step and decrease with each unsuccessful one. Accordingly, the last misclassified instance should have the largest weight. Therefore, we develop a new metric that involves the prioritization difficulty, called ***W**eighted **F**ault **D**etection **R**atio* (WFDR).

We compute the WFDR percentage by the following equation:

$$f(x) = \begin{cases} 1 & \text{if x is misclassified} \\ 0 & \text{otherwise} \end{cases}$$
$$\text{Actual} = \sum_{i=1}^{m} f(x_i) * \underbrace{\left[1 - \frac{m - d_{i-1}}{n-(i-1)}\right]}_{\text{Weights}} \quad (4)$$
$$\text{Ideal} = \sum_{i=1}^{m} \frac{n-m}{n-i+1} \quad (5)$$
$$\text{WFDR} = \frac{Actual}{Ideal} * 100\% \quad (6)$$

where m is the labeling budget, which equals the total number of faults, and n is the size of the test dataset. Also, d_i is the total number of the detected faults within the labeling budget (i). Under the ideal case, all faults are detected within the budget (i), hence, all $f(x_i) = 1$. Initially, no faults are detected, i.e., $d_0 = 0$.

Example 5: We revisit example 3 to evaluate datasets A and B by WFDR. We have m = 10, and n = 100. The prioritized datasets are A = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0], and dataset B =[1, 0, 0, 1, 1, 1, 1, 1, 1, 1]. The WFDR evaluates both approaches as follows:

$$Actual_A = \left[1 - \tfrac{10}{100}\right] + \left[1 - \tfrac{9}{99}\right] + \left[1 - \tfrac{8}{98}\right] + ... + 0 * \left[1 - \tfrac{4}{93}\right] + 0 * \left[1 - \tfrac{4}{92}\right] + 0 * \left[1 - \tfrac{4}{91}\right] = 5.54$$

$$Ideal_A = 9.433, \qquad WFDR_A = 58.73\%$$

$$Actual_B = 7.456,\ Ideal_B = 9.433,\ WFDR_B = 79.05\%$$

Algorithm performance in dataset A reduces from 88.74% by ATRC and 81.82% by RAUC to 58.73% by WFDR. Likewise, the performance in dataset B increases from 69.09% by RAUC and 66.42% by ATRC to 79.05% by WFDR.

Property 1: WFDR approaches its upper bound limit (FDRO) when the misclassification ratio is very small.

Example 5 demonstrates that $FDRO_A$ is 60% and $FDRO_B$ is 80%. We see that the WFDR values are close to the FDRO values in datasets A and B, i.e., 58.78% and 79.05%, respectively. On the other hand, other metrics inaccurately over-evaluate the algorithm performance larger than the fault detection ratio. For example, the ATRC value in dataset A is 88.74% larger than 80%, and the RAUC value in dataset B is 69.09% larger than 60%.

In addition, we revisit example 3 to study how the misclassification ratio strongly affects the WFDR percentage. Figure 2 shows that the WFDR percentage approaches the FDRO percentage exponentially as the test dataset size increases from 14 (the minimum size of dataset A) to 200 and from 12 (the minimum size of dataset B) to 200. We conclude that as the dataset gets larger, the weights increase and the prioritization difficulty becomes higher accordingly. In other words, each weight approaches 1 in eq. 4, and the sum of the weights is roughly the number of detected faults. In the ideal case, all $f(x_i) = 1$ in eq. 5, thus, the sum of all weights is roughly the total number of faults in the test dataset. From eq. 6, we obtain a WFDR value close to the FDRO, as shown in Fig. 2.

Property 2: If two prioritized datasets have the same FDRO, the one with a higher FDRE has a greater WFDR.

The property 2 holds when the misclassification ratio is a large value. When the misclassification ratio is small, the difference between the WFDR values of the two datasets diminishes drastically. For instance, if there are two prioritized datasets: A=[1,1,1,1,1,1,1,0,0,0] and B=[1,0,0,0,1,1,1,1,1,1] and the total number of faults in both datasets is constant (10). Figure 3 shows that dataset A significantly outperforms dataset B under small sizes of the test dataset, i.e., large misclassification ratios. The difference between the WFDRs of A & B reduces exponentially as the size of the test dataset increases. We justify property 2 by the weights change in the WFDR equation. The test dataset contains 13 instances: 10 misclassified and 3 incorrectly classified. We compute the WFDR value for dataset B by the following equation:

$$Actual_B = \left[1 - \tfrac{10}{13}\right] + 0 * \left[1 - \tfrac{9}{12}\right] + 0 * \left[1 - \tfrac{9}{11}\right] + ... + \left[1 - \tfrac{9}{9}\right] + \left[1 - \tfrac{8}{8}\right] + ... + \left[1 - \tfrac{4}{4}\right] = 0.231$$

$$Ideal_B = 4.04, \qquad WFDR_B = 5.72\%$$

$$Actual_A = 2.19, Ideal_A = 4.04, WFDR_A = 54.21\%$$

Since the algorithm in dataset B prioritizes all correctly classified instances, the remaining in the 13-test dataset has to be misclassified instances. Thus, there is no need for prioritization as the difficulty is zero, i.e., the weights (shown in blue) are zero. In dataset A, as the algorithm prioritizes the misclassified instances successfully, weights get larger.

From Fig. 3, we can conclude that when the size of the test dataset size is:

- small (e.g., between 13 and 40): the misclassification ratio is large, and the weights are small. Hence, the WFDR metric evaluates the algorithms according to the FDRE since both FDROs are the same.
- medium (e.g., 40 and 100): the misclassification ratio tends to be low, and the weights and the difficulty get larger. Hence, there is much importance for FDRO.
- large (more than 100): in both datasets, the misclassification ratio is very small, and the difficulty is very high. Since both datasets have almost the same weights and difficulty, the WFDR evaluates the algorithms according to the FDRO rather than the FDRE.

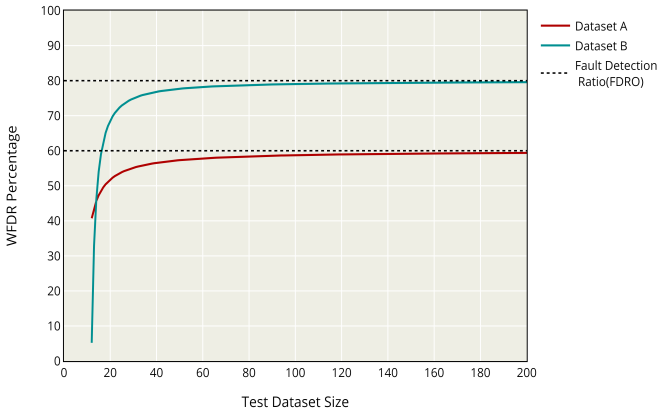

Fig. 2: The effect of the dataset size on the WFDR under different FDROs.

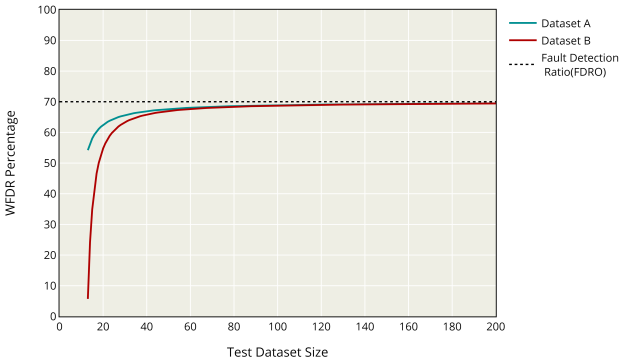

Fig. 3: The effect of the dataset size on the WFDR under similar FDROs.

D. Severe Faults Detection Rate

The existing algorithms prioritize all misclassified instances equally. In situations where the budget is limited or when safety and security are critical, it is insufficient to prioritize all misclassified instances similarly. Thus, the algorithm should consider the *severity* when prioritizing highly severe instances over other misclassified ones.

We estimate the severity level by the potential harm that can occur when the neural network misclassifies a particular instance. As a result, we quantify the severity by the prediction probability. For example, instances with a low-probability prediction of less than 50% are low-severity instances. Accordingly, we should plan safety precautions before model deployment to prevent damaging consequences. Contrarily, instances with a high prediction probability of more than 80% are considered highly severe. No proactive actions are taken since intelligent systems heavily rely on high-probability predictions.

Figure 4 illustrates some highly-severe examples from the CIFAR dataset. These instances reveal the main weaknesses of the trained neural network. As corrective actions, we should retrain the neural network with the following: a) boats with mainsail reflection, b) dogs at different zoom levels, and c) planes in various positions, not only flying.

In severity prioritization, algorithms should prioritize high-severity instances at the top of the list. In this context, the order among misclassified instances is more important. For example, prioritizing low-severity examples at the beginning negatively impacts the performance of the algorithm. The penalty is greater when the algorithm prioritizes correctly classified ones, which are completely safe. Therefore, the rate of prioritization is more important than the ratio.

An ideal list **A** has all misclassified instances in descending order according to the prediction probability. An algorithm prioritizes instances in a specific order in list **B**. To evaluate list **B** against **A**, we develop a new metric, namely ***S**evere **F**ault **D**etection **R**ate* (SFDR). We

(a) label: ship
predicted: airplane
probability: 99.99%

(b) label: dog
predicted: horse
probability: 99.99%

(c) label: airplane
predicted: vehicle
probability: 70.70%

Fig. 4: High severe images in the CIFAR dataset.

compute the SFDR percentage by the following equation:

$$f(x) = \begin{cases} 1 & \text{if x is correctly classified} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{SFDR} = \frac{1}{m} \sum_{i=1}^{m} \gamma^{f(x_i)} * \frac{\left|\mathbf{A}_{[0:i]} \cap \mathbf{B}_{[0:i]}\right|}{i} * 100\% \qquad (7)$$

where γ ranges $]0, 1[$, and γ controls the degree of the penalty. Moreover, the γ value and the penalty degree are inversely proportional. Thus, we select $\gamma = 0.5$ in our evaluations. Furthermore, the SFDR percentage ranges between 0% and 100% from worst to optimal performance, respectively. Lastly, the intersection operator allows for duplicated severe cases because some misclassified instances have the same severity degree. More importantly, the SFDR is a top-weighted metric, i.e., the metric imposes more weight on the top of the prioritized list. Thus, the cost of incorrect prioritization decays gradually along with the prioritized list.

We partition equally the degree of severity, i.e., the prediction probability, into 10 levels between 0% and 100%. For example, misclassified instances with a prediction probability of 90s% should be at the head of the prioritization list. On the other side, all correctly classified instances and misclassified instances with prediction probability between 1% and 10% should be at the tail of the list. It is worth noting that the prioritization among all misclassified instances of the same severity level does not matter, as illustrated by the following examples.

Example 6: An ideal list **A** = [99, 91, 85, 83, 64, 24], **A** contains all misclassified instances in a small test dataset, and the prioritized list **B** = [99, 0, 91, 83, 64, 0]. The labeling budget $m = 10$ and $\gamma = 0.5$. Since the severity of correctly classified instances is zero, we replace the prediction probability with zero. Let $\mathbf{x}_i$ be $\left|\mathbf{A}_{[0:i]} \cap \mathbf{B}_{[0:i]}\right|$.

The first step to evaluate the prioritization list by SFDR metric is to encode the prediction probabilities into severity levels between 1 and 10, where 10 is the highest degree of severity (prediction probability between 90% and 100%) and 1 is the lowest degree of severity. More particularly, we replace all prediction probabilities with the corresponding level of severity. Thus, **A** becomes [10, 10, 9, 9, 7, 3] and **B** becomes [10,0,10,9,7,0].

TABLE I: The SFDR metric evaluates the list **B**.

i	$\mathbf{A}_{[0:i]}$	$\mathbf{B}_{[0:i]}$	$\mathbf{x}_i$	$f(x_i)$	$\mathbf{y}_i$
1	[10]	[10]	1	0	1.00
2	[10, 10]	[10, 0]	1	1	0.25
3	[10, 10, 9]	[10, 0, 10]	2	0	0.67
4	[10, 10, 9, 9]	[10, 0, 10, 9]	3	0	0.75
5	[10, 10, 9, 9, 7]	[10, 0, 10, 9, 7]	4	0	0.80
6	[10, 10, 9, 9, 7, 3]	[10, 0, 10, 9, 7, 0]	4	1	0.33
$SFDR = \frac{1}{m}\sum_{i=1}^{m} \mathbf{y}_i * 100\%$					63.33%

Table I shows the step-by-step computation for the SFDR percentage. If the prioritized instance is correctly classified, the $f(x_i)$ value is one, and γ is 0.5. Thus, the SDFR value drastically declines when i is 2, and the value of $\mathbf{y}_i = \gamma^{f(x_i)} * (\mathbf{x}_i/i)$ largely decreases from 1 to 0.25. When i is 3, the intersection between the two lists allows duplication in the severity degree (10).

Table II demonstrates how the SFDR metric evaluates effectively different prioritization lists against **A** = [99, 90, 88, 81, 75, 70, 69, 62]. For example, in list **B**, the second incorrect prioritization heavily penalizes the performance with a significant drop from 100% to 80.09%. Moreover, the performance in **C** is worse than in **B** since the algorithm prioritizes falsely the last four instances that are non-fault instances. In scenarios **D** and **E**, the first four instances are incorrectly prioritized. The main difference between the two scenarios is the type of instances: misclassified instances with lower severity in scenario **D** and correctly classified instances in scenario **E**. As a result, there is an 18.28% drop in the SFDR percentage between the two scenarios. Note that calculations are not included for the sake of brevity, and only SFDR values are presented.

TABLE II: SFDR evaluates different prioritization lists.

Name	List	SFDR (%)
B	[90, 88, 81, 75, 70, 69, 62, 99]	80.09
C	[99, 90, 88, 81, 0, 0, 0, 0]	65.86
D	[75, 70, 69, 62, 99, 90, 88, 81]	36.55
E	[0, 0, 0, 0, 99, 90, 88, 81]	18.27

IV. Experimental Setup

The experiments were conducted using a machine with an Nvidia K80 GPU and 12 GB of RAM, implemented using the PyTorch v1.9.0 framework. Table III provides a summary of the details of the main experiments. We detail our main setup as follows:

- **Datasets**: MNIST [12], Fashion-MNIST [13], CIFAR10 [14], SVHN [15].
- **Pretrained Model**: ResNet18 [16], GoogLeNet [17], ResNet34 [16], ResNet50 [16], ResNet101 [16], ResNet152 [16], and EfficientNet-B0 [18].
- **Prioritization Algorithms**: DeepGini, Neurons pattern, and DeepAbstraction.

TABLE III: Details of the datasets and pretrained models.

Exp ID	Dataset	Training Dataset	Test Dataset	Pretained Model	Training Acc. (%)	Test Acc. (%)
Exp 1	CIFAR-10	50,000	10,000	Efficient-B0	94.95	92.86
Exp 2	CIFAR-10	50,000	10,000	ResNet101	88.83	86.97
Exp 3	F-MNIST	60000	10000	Efficient-B0	94.94	94.17
Exp 4	F-MNIST	60,000	10,000	ResNet50	93.11	91.12
Exp 5	MNIST	60,000	10,000	ResNet18	99.36	99.16
Exp 6	MNIST	60,000	10,000	ResNet34	99.29	98.84
Exp 7	SVHN	73,257	26,032	GoogLeNet	95.51	95.07
Exp 8	SVHN	73,257	26,032	ResNet152	94.63	94.10

- **Research Questions**:
  - ❶ **(Metrics Effectiveness)**: How effective are the existing and proposed metrics in evaluating the prioritization algorithms?
  - ❷ **(Algorithms Effectiveness)**: How effective are the existing algorithms evaluated by the WFDR and SFDR metrics?
  - ❸ **(Severity Distribution)**: What is the distribution of highly severe instances among the widely used benchmarks?

V. Experimental Evaluation

This section addresses the research questions outlined in Sec. IV. The results are in the repository, which will be publicly available upon the paper's acceptance.

A. RQ1: Metrics Effectiveness

Figure 5 shows that the APFD metric overestimates the performance of all algorithms. For example, the APFD values for all experiments involving the DeepGini algorithm are between 94.14% and 99.64%. The APFD values are significantly larger than the other metrics (RAUC, ATRC) with differences up to 74%. Additionally, a comparison of APFD and ARTC was conducted to understand the impact of the labeling budget and found that APFD values are greater than ARTC values with significant differences, reaching up to 47.19%, 82.91%, and 27.26% for each algorithm (a,b, and c). Since APFD does not consider the misclassification ratio, it incorrectly exceeds the evaluation of WFDR by a considerable margin of more than 70%.

RAUC and ARTC are not weighted metrics. Thus, they overestimate the performance, which exceeds the FDRO. For example, when comparing the evaluations of RAUC and WFDR for the DeepAbstraction algorithm among all experiments, the difference reaches 13.89%. Similarly, the difference between the evaluations of ATRC and WFDR reaches 20.04%, as shown in Fig.5. Furthermore, RAUC and ARTC values among all experiments exceed the FDRO shown in Table IV. On the other hand, all values of the WFDR metric are either below or close to FDRO. For instance, in Exp 8, the FDRO of the DeepGini algorithm is 47.33%, while the RAUC, ARTC, and WFDR values are 53.05%, 57.08%, and 46.55%, respectively.

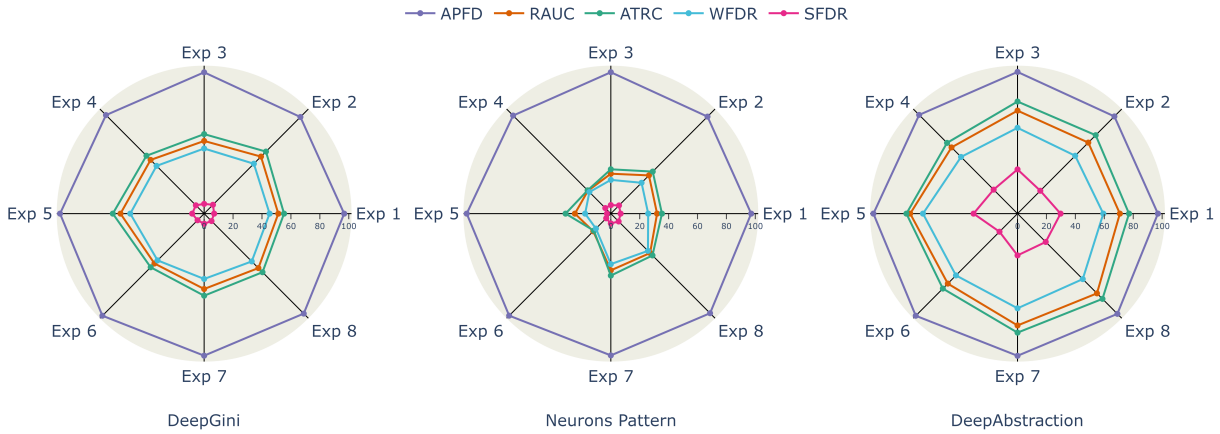

Fig. 5: Evaluations of the different test prioritization metrics on various experiments.

TABLE IV: Fault detection ratio for several algorithms.

Exp ID	No. Bugs	Fault Detection Ratio (%)		
		DeepGini	Neurons Pattern	DeepAbstraction
Exp 1	714	46.22	26.61	60.22
EXP 2	1303	50.50	31.70	58.17
Exp 3	583	45.63	23.84	59.86
Exp 4	888	47.75	21.85	56.42
Exp 5	84	51.19	17.86	65.48
Exp 6	116	45.69	14.66	60.34
Exp 7	1284	45.64	35.44	65.81
Exp 8	1538	47.33	37.06	64.50

TABLE V: Fault Detection Ratio (%) for different algorithms according to the severity levels.

Exp ID	FDRO of Severe Bugs				FDRO of Moderate Bugs			
	No. Bugs	Deep Gini	Pattern Algor.	Deep Abst.	No. Bugs	Deep Gini	Pattern Algor.	Deep Abst.
1	315	0.00	13.65	47.62	312	77.88	31.73	55.45
2	325	0.00	10.15	40.00	656	51.22	29.42	45.58
3	233	0.00	13.30	54.51	285	70.53	23.86	50.53
4	262	0.00	11.07	72.52	504	59.92	19.44	44.44
5	36	0.00	5.56	69.44	38	86.84	15.79	18.42
6	49	0.00	10.20	63.27	50	72.00	12.00	44.00
7	409	0.00	3.91	74.82	534	45.88	33.90	41.01
8	331	0.00	3.02	78.55	662	29.76	27.04	36.71

Lastly, Fig. 5 shows that SFDR evaluates the DeepGini algorithm as better than the neurons pattern algorithm. Table V shows the FDRO of two levels of severity: high level in which the prediction probability is greater than or equal to 80%, and moderate level in which the probability is between 50% and 80%. For instance, in Exp 5, 84 misclassified instances have different levels of severity: 36-high, 38-moderate, and 10-low. DeepGini prioritizes 0-high, and 33-moderate severity instances, while the neurons pattern algorithm prioritizes 2-high and 6-moderate severity instances. Hence, the SFDR values for DeepGini and neurons pattern algorithms are 8.3%, and 2.7%, respectively.

We can also observe that DeepAbstraction outperforms DeepGini, as the former algorithm prioritizes 25 highly severe instances, while the latter fails to prioritize any such instances. Nonetheless, DeepGini and DeepAbstraction prioritize 86.84%, and 18.42% of the moderately severe examples, respectively. As a result, the SFDR metric evaluates DeepAbstraction with (21.28%) as better than DeepGini with (8.05%). To sum up, the SFDR metric evaluation is consistent with the results in Table V.

RQ 1 Answer :

The existing metrics are ineffective and also over-evaluate the performance more than the FDRO. On the other hand, the experiments show the validity of WFDR and SFDR evaluations.

B. RQ2: Algorithms Effectiveness

The answer to **RQ1** confirms that the proposed metrics effectively evaluate the performance of algorithms.

The evaluation of the WFDR metric shows that Deepabstraction performs significantly better than other algorithms in all experiments, as demonstrated in Fig. 6. For example, Deepabstraction outperforms DeepGini by a significant margin (up to 20.23%) as shown in Fig. 6. Since the WFDR metric relies heavily on the FDRO, the WFDR metric indicates that DeepGini performs better than the neurons pattern algorithm, which is consistent with the FDRO values in Table IV.

Figure 6 illustrates that all algorithms perform poorly in prioritizing highly-severe instances, with SFDR values at most 30%. However, DeepAbstraction performs significantly better than other algorithms (DeepGini, Neurons Pattern), with margins of 23.8% and 27.7%, respectively.

The Gini score is inversely proportional to the certainty of the model, which is estimated by the prediction probability. DeepGini prioritizes high Gini instances with low certainty. i.e., low probability prediction. In other orders, low Gini instances with high prediction

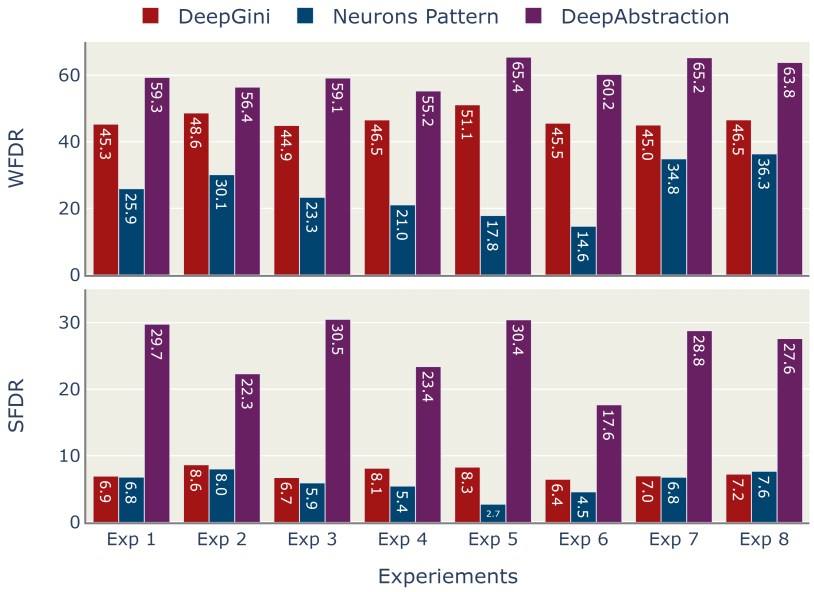

Fig. 6: WFDR & SFDR evaluate different algorithms.

probability (high severity) are at the bottom of the priority list, resulting in poor performance in the severity prioritization. Table V confirms this by showing that DeepGini does not prioritize any highly severe instances with prediction probability greater than 80%. But DeepGini prioritizes many moderate-severity instances.

The neurons pattern algorithm heavily relies on the Familiarity score (FD+) to measure the conformance degree between the established pattern during the training and the test instance. High-severity instances have a deceptive similarity to the established pattern and thus have a high FD+ score, resulting to be at the bottom of the prioritization list.

Lastly, DeepAbstraction uses monitors to prioritize all rejected test instances at the beginning of the list. However, using the Gini score to prioritize these instances degrades the performance of DeepAbstraction as shown in Fig. 6. Table V confirms the good performance of DeepAbstraction by detecting many high and moderate-severity instances. Since the rate matters more than the ratio in the severity prioritization, the performance of DeepAbstraction cannot exceed 30%.

RQ 2 Answer :

The performance of all studied algorithms needs to be improved, with poor WFDR of less than 70% and SFDR values of no more than 30.5%. However, DeepAbstraction algorithm shows significant improvement compared to the others in both measures.

C. RQ3: Severity Distribution

We investigate the significance of severity prioritization. In this regard, we evaluate the ratio of high-severity instances in relation to the overall count of misclassified instances. Figure 7 provides a visual representation of this distribution, showing that high-severity instances are the most prevalent among the five experiments.

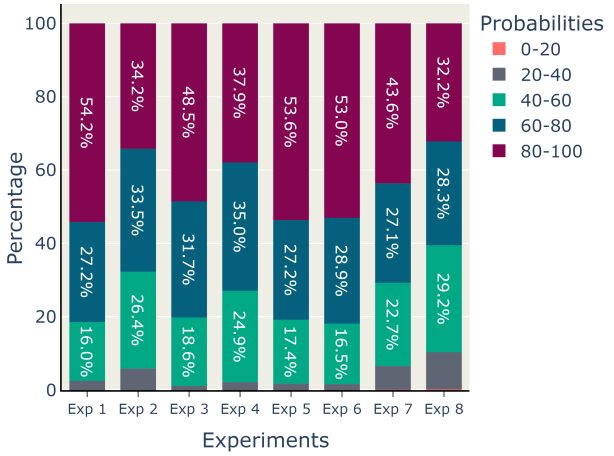

Fig. 7: The distribution of different levels of severity.

Moreover, the data shows that a considerable percentage, over 20% in fact, of instances in every benchmark, are of high severity. These figures underscore the prevalence of high-severity instances in our datasets, thus highlighting the depth of the problem. As a result, there is a pressing need to develop new algorithms that prioritize only highly-severe instances at the top of the priority list.

RQ3 Answer :

Many instances in all benchmarks are highly severe, highlighting the need for new prioritization algorithms that specifically address the severity issue.

VI. Conclusion and Future Work

This paper confirms that commonly used metrics for evaluating prioritization algorithms, such as APFD, RAUC, and ATRC, are ineffective. For instance, the APFD metric neglects the labeling cost, leading to over-evaluating the performance. Also, the RAUC and ATRC metrics mainly evaluate the algorithms based on FDRE instead of FDRO. All existing metrics also neglect the misclassification ratio that represents the difficulty of prioritization. Therefore, we developed the WFDR metric, which is a weighted metric that assigns weights to each prioritization step according to the prioritization difficulty. The experiments also reveal that highly severe test instances make up a large portion of all studied datasets, highlighting the importance of severity prioritization. Accordingly, we developed the SFDR metric, which is a top-weighted metric that evaluates the algorithm more heavily on the top of the list. Eventually, the empirical experiments validate the effectiveness of the WFDR and SFDR metrics and the poor performance of the studied algorithms.

Future work should aim to develop new prioritization algorithms with higher WFDR values. Moreover, more focus should be on algorithms that prioritize high-severity instances quickly. Further studies should also examine the applicability of proposed metrics to evaluate active learning algorithms. Lastly, more work should investigate why the neural network misclassifies the highly severe instances with the aid of any techniques of explainable AI (XAI) such as LIME [19], DeepLIFT[20], and SHAP[21].

References

[1] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020.

[2] M. Weiss and P. Tonella, "Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study)," *arXiv preprint arXiv:2205.00664*, 2022.

[3] R. Yan, Y. Chen, H. Gao, and J. Yan, "Test case prioritization with neuron valuation based pattern," *Science of Computer Programming*, vol. 215, p. 102761, 2022.

[4] Y. Li, M. Li, Q. Lai, Y. Liu, and Q. Xu, "Testrank: Bringing order into unlabeled test instances for deep learning tasks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[5] Y. Tao, C. Tao, H. Guo, and B. Li, "Tpfl: Test input prioritization for deep neural networks based on fault localization," in *International Conference on Advanced Data Mining and Applications*, pp. 368–383, Springer, 2022.

[6] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 397–409, IEEE, 2021.

[7] J. Chen, J. Ge, and H. Zheng, "Actgraph: Prioritization of test cases based on deep neural network activation graph," *arXiv preprint arXiv:2211.00273*, 2022.

[8] Z. Wei, H. Wang, I. Ashraf, and W. Chan, "Predictive mutation analysis of test case prioritization for deep neural networks," in *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*, pp. 682–693, IEEE, 2022.

[9] H. Al-Qadasi, C. Wu, Y. Falcone, and S. Bensalem, "Deepabstraction: 2-level prioritization for unlabeled test inputs in deep neural networks," in *2022 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 64–71, IEEE, 2022.

[10] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, pp. 102–112, 2000.

[11] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Input prioritization for testing neural networks," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 63–70, IEEE, 2019.

[12] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[13] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[14] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[15] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS*, 2011.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[18] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning (ICML)*, 2019.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144, 2016.

[20] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*, pp. 3145–3153, PMLR, 2017.

[21] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.