PriCod: Prioritizing Test Inputs for Compressed Deep Neural Networks

YINGHUA LI, University of Luxembourg, Luxembourg
XUEQI DANG*, University of Luxembourg, Luxembourg
JACQUES KLEIN, University of Luxembourg, Luxembourg
YVES LE TRAON, University of Luxembourg, Luxembourg
TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg

The widespread adoption of deep neural networks (DNNs) has brought remarkable advances in machine learning. However, the computational and memory demands of complex DNNs hinder their deployment in resource-constrained environments. To address this challenge, compressed DNN models have emerged, offering a compromise between efficiency and accuracy. Nonetheless, assessing the performance of these compressed models can demand extensive testing, typically requiring high manual labeling costs, rendering the process resource-intensive and time-consuming. To mitigate these challenges, test input prioritization has emerged as a promising technique aimed at reducing labeling costs by prioritizing inputs that are more likely to be misclassified. This enables the early identification of bug-revealing tests with reduced time and manual labeling effort. In this paper, we propose PriCod, a novel test prioritization approach designed for compressed DNNs. PriCod leverages the behavior disparities caused by model compression, along with the embeddings of test inputs, to effectively prioritize potentially misclassified tests. It operates on the premises that significant behavior disparities between the models indicate potential misclassifications and that inputs near decision boundaries are more likely to be misclassified. To this end, PriCod generates two types of features for each test input (i.e., deviation features and embedding features) to capture the prediction deviation caused by model compression and the proximity to decision boundaries, respectively. By combining these features, PriCod predicts the probability of misclassification for each test, ranking tests accordingly. We conduct an extensive study to evaluate the effectiveness of PriCod, comparing it with multiple test prioritization approaches. The experimental results demonstrate the effectiveness of PriCod, with average improvements of 7.43%~55.89% on natural test inputs, 7.92%~52.91% on noisy test inputs, and 7.03%~51.59% on adversarial test inputs, compared with existing test prioritization approaches.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**; • **Computer systems organization** → *Neural networks.*

Additional Key Words and Phrases: Test Input Prioritization, Deep Neural Network, Learning to Rank, Labeling

1 INTRODUCTION

The widespread use of deep neural networks (DNNs) has brought significant advancements to machine learning in areas like computer vision [11, 95], autonomous vehicles [41, 109], and recommendation systems [4]. However, the increasingly complex DNNs require substantial computational resources and memory, limiting their practical deployment in resource-constrained environments, such as edge devices. To tackle these challenges, the research

*Corresponding author.

Authors' addresses: Yinghua Li, yinghua.li@uni.lu, University of Luxembourg, Luxembourg; Xueqi Dang, xueqi.dang@uni.lu, University of Luxembourg, Luxembourg; Jacques Klein, jacques.klein@uni.lu, University of Luxembourg, Luxembourg; Yves Le Traon, yves.letraon@uni.lu, University of Luxembourg, Luxembourg; Tegawendé F. Bissyandé, tegawende.bissyande@uni.lu, University of Luxembourg, Luxembourg.

ACM 1557-7392/2025/4-ART
https://doi.org/10.1145/3730435

community has focused on the development of compressed DNN models that strike a balance between computational efficiency and model accuracy. Compressed DNN models, essentially scaled-down neural networks, are designed to sustain predictive accuracy while keeping computational requirements to a minimum. A multitude of compression techniques, including quantization [82], have emerged as valuable tools for reducing the size and computational load of DNNs while safeguarding their predictive prowess. Consequently, the evaluation and validation of compressed DNNs have become increasingly crucial to ensure their performance.

Existing studies [64–66] mentioned that, when evaluating compressed DNN models, labeling new test cases is necessary. However, labelling test inputs for compressed DNN models faces a central challenge: the high labelling cost issue. This challenge arises for two main reasons. Firstly, the scale of the test set can be extensive. Secondly, manual labeling is still the mainstream approach, requiring the participation of multiple annotators to guarantee the accuracy of the labeling process for each test input. A highly appealing solution to this challenge is test input prioritization. This technique prioritizes test inputs that are more likely to be misclassified when resources and time are limited for manual labeling. By prioritizing such potentially misclassified test inputs, testers can allocate labeling resources more effectively and thus enhance debugging efficiency.

In the literature [29, 97, 99], several test prioritization methods have been proposed for traditional DNNs, which can be broadly classified into three categories: coverage-based [60, 79], confidence-based [29, 62, 99], and mutation-based methods [97]. Coverage-based approaches prioritize test inputs based on the extent of neuron coverage within DNNs. Conversely, confidence-based methods aim to identify potential misclassifications by quantifying the classifier's output confidence for each test case. Notably, the DeepGini approach by Feng et al. [29] utilizes the Gini score as a confidence metric for effective test prioritization. More recently, Weiss et al. [99] conducted a comprehensive study, which included an evaluation of several confidence-based metrics, such as Vanilla Softmax, Prediction-Confidence Score (PCS), and Entropy. Mutation-based techniques propose various mutation operations and employ the modified results to prioritize test cases.

Despite the goal for both compressed and traditional DNN models' test prioritization methods being to select potentially misclassified test inputs, test prioritization for compressed models has its unique specialness and challenges compared to that for traditional DNN models, which we present as follows:

- **Challenge of Obtaining Internal Information** Compared to the test prioritization of traditional DNN models, when performing test prioritization on compressed DNN models, it is challenging to obtain model internal information, such as gradient information. This is because compression techniques can alter the internal structure of the model, making internal information, which is easily accessible in uncompressed models, difficult to extract in compressed models.
- **Diverse Operating Environments** Additionally, the primary applications of compressed models are usually for deployment on mobile devices, where the images input into the model are typically photos taken by mobile cameras. These inputs can contain noises, such as raindrops, unusual exposure levels, and various shooting angles. Therefore, the test prioritization methods for compressed models need to consider these diverse operating environments to ensure effectiveness under various conditions.

Moreover, for the specific existing test prioritization methods mentioned above, when they are applied to compressed models, they have the following limitations: Firstly, previous research [29] has demonstrated that coverage-based methods are less effective and more time-consuming when compared to confidence-based approaches. Secondly, the mutation-based test prioritization approach, PRIMA [97], cannot be applied to compressed DNN models because the model mutation operators of PRIMA are not applicable to them. This limitation arises from the fact that the architectures and gradients of compressed models are typically unavailable [93]. Furthermore, when employing confidence-based test prioritization approaches for compressed DNN models, these methods treat the compressed DNN models as black boxes and ignore the information regarding deviations before and after model compression when conducting test prioritization.

In this paper, we propose PriCod (**Pri**oritizing test inputs for **Co**mpressed **D**NN models), a test prioritization approach specifically tailored for compressed DNN models. PriCod leverages the deviation between the original model and compressed DNN models, along with the embedding information of test inputs, to perform test prioritization. The fundamental principle underlying PriCod is twofold:

- **Premise 1:** For a given test, if the prediction behavior between the compressed DNN model and the original model shows a large deviation, it suggests that, for this input, the compressed model is more likely to make a prediction different from that of the original model. This test is considered more likely to reveal bugs in the compressed model. We validated premise 1 through a targeted preliminary study. Details can be found in Section 3.1.
- **Premise 2:** Test inputs that are located closer to the decision boundary of the model are more likely to be misclassified. This premise has been previously established in prior research [61].

Building upon the aforementioned premises, PriCod generates two distinct types of features: deviation features and embedding features. To ensure a comprehensive representation of deviation information, PriCod employs a set of 17 strategies for generating deviation features. For each test input, PriCod combines these two feature types to predict the probability of the test being misclassified by compressed DNN models. Below, we provide an overview of the two types of features and elaborate on how PriCod utilizes them to achieve effective test prioritization.

- **Deviation Features** Deviation features are specifically designed to capture the impact of model compression on test inputs. They quantify the disparities in predictions between the original DNN model and the compressed DNN model for each test input. We generated 17 types of deviation features, such as Cosine Similarity [71] and Manhattan Distance [14], with the aim of providing a more comprehensive quantification of the disparity.
- **Embedding Features** Embedding Features encapsulate the representative information within each test input. Through the process of mapping a test input to a vector in space, PriCod aims to indirectly unveil the proximity between the test input and the decision boundary.

For each test instance, PriCod integrates the aforementioned two types of features to derive the ultimate feature vector. Using this vector, PriCod learns the misclassification probability of this test. Ultimately, PriCod ranks all tests within a test set based on their misclassification probabilities.

PriCod demonstrates novelty compared to existing test prioritization methods. Specifically, PriCod first leverages the behavior disparities caused by model compression for test prioritization. These behavior disparities are a unique characteristic of compressed DNN models. Although existing studies proposed some mutation-based test prioritization methods, notably PRIMA [97] and GraphPrior [17], PriCod differs from PRIMA and GraphPrior in several aspects:

- **Not using mutation testing for test prioritization** The core of test prioritization for PRIMA and GraphPrior is based on mutation testing, which focuses on proposing new mutation operators and generating mutation features for test prioritization. In contrast, PriCod does not use mutation testing for test prioritization. Specifically, PriCod does not involve any mutation testing operations and does not use any mutation operators for test prioritization. Importantly, PriCod leverages the behavior disparities caused by model compression for test prioritization. These behavior disparities are a unique characteristic of compressed models. We are the first to introduce this characteristic for test prioritization.
- **Applicability to compressed DNN models** PRIMA and GraphPrior cannot be applied to compressed DNN models. In contrast, PriCod is specifically designed for compressed DNN models. The reasons why PRIMA and GraphPrior cannot be applied to compressed models are as follows: 1) PRIMA requires access to internal model information of DNNs, such as gradient information. However, currently, to the best of our knowledge, accessing internal information of compressed models is not feasible. This leads to PRIMA not being suitable for compressed DNN models. 2) GraphPrior is designed for GNNs. Specifically, its mutation operators are

specifically designed for GNN models and datasets with graph structures. Hence, GraphPrior is not applicable to compressed DNN models.

- **Utilizing different feature fusion techniques** In PriCod, we conducted an analysis of feature fusion techniques, which GraphPrior and PRIMA do not include. Specifically, we utilized a total of four feature fusion techniques (cf. Section 5.4) and compared the effectiveness of PriCod when utilizing different feature fusion techniques.

PriCod demonstrates its wide applicability across various contexts of compressed DNNs. For instance, in the context of medical image diagnosis, hospitals utilize DNN models to diagnose lung diseases in X-ray chest images. To overcome storage and computational limitations, they opt for compressed DNN models. However, this compression process carries the risk of accuracy loss, which can lead to treatment delays, missed early interventions, and patient health deterioration. PriCod can be used to effectively identify images at higher risk of misclassification by the compressed DNN model. By prioritizing these samples for screening, it can reduce the risk of misdiagnosis and enhance the reliability of the diagnostic model.

We conducted an extensive study to assess PriCod's performance, utilizing a dataset comprising 182 subjects (paired datasets and compressed DNN models). The evaluation covered a diverse range of test inputs, including natural data, noisy data, and adversarial data. Additionally, we meticulously selected a set of test prioritization approaches for comparison, which have previously proven effective in existing studies [29, 99]. Furthermore, we included random selection as the baseline approach. Our experimental results highlight PriCod's superior performance compared to existing methods. When applied to natural test inputs, PriCod demonstrates an average improvement ranging from 7.43% to 55.89%. For noisy and adversarial test inputs, it exhibits an average improvement ranging from 7.92% to 52.91% and from 7.03% to 51.59%, respectively. We publish our dataset, results, and tools to the community on Github[1].

Our work has the following major contributions:

- **Approach** We propose PriCod, a novel test prioritization approach designed specifically for compressed DNN models. Our approach leverages the discrepancies in predictions before and after model compression, as well as the embedding features of tests, to guide test prioritization.
- **Study** To evaluate PriCod, we conduct an extensive study involving 182 subjects, encompassing natural, noisy, and adversarial datasets. Within this study, we systematically evaluate PriCod in comparison to multiple test prioritization approaches. Our experimental results demonstrate the effectiveness of PriCod.
- **Performance Analysis** We assess the contributions of various feature types to the performance of PriCod. Additionally, we analyze the impact of feature fusion techniques on PriCod's effectiveness. Furthermore, we investigate the relationship between the misclassification probability and the deviated behaviors.

2 BACKGROUND

2.1 DNNs and DNN testing

Classification deep neural networks (DNNs) [34] serve as the core of numerous deep learning (DL) systems. Classification refers to the task of categorizing input data into different classes [111], such as identifying objects in images or categorizing emails as spam or non-spam. DNNs can perform the classification task by learning mappings from input data to specific categories. They adjust the weights and parameters within the network through training data, enabling the network to automatically capture patterns and features in the data, thereby achieving accurate classification.

A DNN consists of multiple layers: an input layer, an output layer, and one or more hidden layers. Each layer is composed of a series of neurons. The input layer is the first layer of the network, responsible for receiving raw

[1]https://github.com/yinghuali/PriCod

data. The output layer is the final layer of the network, generating the final prediction results. Hidden layers lie between the input and output layers. They transmit information and perform feature extraction within the network. Each hidden layer comprises multiple neurons that combine and transmit information through weights and activation functions. In the context of DNNs, neurons are the fundamental computational units. Neurons in hidden and output layers are interconnected with all neurons in the preceding layer through weighted edges. The weights of these edges are automatically learned during a training process using a large dataset with labeled training examples. Following training, a DNN can autonomously classify input samples, such as images, into their respective categories. For example, within the framework of a DNN model designed for animal classification tasks, it can differentiate between various types of animals in images, precisely labeling whether it is a cat, dog, or bird.

Ensuring the quality and reliability of DNN models is of paramount importance. DNN testing has emerged as a widely adopted approach to achieve this goal [1, 17, 42, 43]. Similar to traditional software systems [36, 76, 77, 107], DNN testing relies on the use of inputs and oracles. In the context of DNN testing, test inputs represent the data that the model is expected to classify. These inputs can take various forms depending on the specific task of the DNN under examination, such as images, natural language, or speech. Test oracles in DNN testing involve manual labeling, wherein human annotators manually assign ground truth labels to each input. By comparing these labeled ground truth labels with the predicted output of the DNN model, it becomes possible to evaluate the model's accuracy in generating the correct output for a given input.

2.2 DNN Model Compression

Model compression has emerged as a promising avenue of research to facilitate the deployment of DNN models [15, 87, 104]. The primary objective of DNN model compression is to minimize the computational and memory requirements of models while maintaining their performance, thus enabling effective deployment in resource-constrained contexts. A variety of techniques have been proposed for compressing DNN models. Among these techniques, Quantization plays a crucial role, which operates by compressing a DNN model through the adjustment of bit numbers allocated to weight representation [113]. In the conventional landscape of DNN models, weights find their common expression in the form of 32-bit floating-point numbers. The primary goal of quantization is to reduce the bit precision of parameters in neural network models, thereby decreasing storage and computational overhead while maintaining optimal model performance. This involves the utilization of reduced bit representations like 8-bit, which in turn substantially curtails the storage requirements of the model. In our study, we primarily employed quantization as the method for model compression.

The currently trending model compression techniques primarily encompass two options: TensorFlow Lite (TFLite) [18] and CoreML [91]. These two compression methodologies have gained extensive traction within the mobile device domain [53]. TFLite, developed by Google, is a deep learning inference framework explicitly designed for mobile and embedded devices. Its standout feature is its highly optimized computational performance, which ensures the efficient execution of trained neural network models on the Android platform [28]. On the other hand, CoreML is Apple's solution for deep learning inference on mobile devices, tailored exclusively for the iOS ecosystem. CoreML leverages the inherent hardware advantages of Apple devices and achieves rapid inference for neural network models through strategic hardware acceleration implementation. In our research, we employed the aforementioned techniques (TFLite and CoreML) to compress the original DNN models into compressed DNN models, aiming to offer a more comprehensive evaluation.

2.3 Confidence-based Test Prioritization for DNNs

Confidence-based methods identify inputs that can potentially expose bugs (i.e., possibly misclassified test inputs) by analyzing the output probabilities of a DNN classifier. One classic confidence-based test prioritization technique is DeepGini, which prioritizes test inputs by calculating Gini scores for each input. These scores measure the model's confidence in classifying each input, thus facilitating test prioritization. More specifically, if a DNN produces similar probabilities for all classes towards a test input, it indicates lower confidence in the classification. As a result, this input will be prioritized higher. DeepGini has demonstrated effectiveness across various prevalent DNN datasets, such as CIFAR10 (color images) [20] and Fashion (fashion product images) [102]. Recently, Weiss *et al.* [99] extensively explored diverse techniques for prioritizing DNN test inputs, encompassing a series of confidence-based approaches, such as Vanilla Softmax, Prediction-Confidence Score (PCS), and Entropy. These metrics have been proven effective in DNN test prioritization. Compared with coverage-based test prioritization methods (e.g., CTM and CAM), which prioritize inputs based on neuron coverage, confidence-based methods offer several distinct advantages.

- **Efficiency** Confidence-based methods require minimal time and computational resources, as they solely rely on statistical computations of confidence levels in the predicted probability vectors from the output softmax layer.
- **Effectiveness** Confidence-based methods have demonstrated higher effectiveness compared to various coverage-based test prioritization techniques.
- **Minimal Need for Intermediate Information** Unlike coverage-based methods, confidence-based approaches do not necessitate the collection of extensive intermediate information to compute coverage rates. Additionally, they enhance security by not requiring an in-depth inspection of the DNN, thereby safeguarding sensitive information embedded within the network.

However, applying confidence-based methods to compressed DNN models comes with a limitation. These methods solely depend on the predictive confidence information of the compressed DNN model without considering the differences between the compressed model and the original model. Our proposed approach, PriCod, incorporates this deviation information in the test prioritization process. We compared PriCod with a set of confidence-based test prioritization techniques and demonstrated that Pricod outperforms all the comparative methods, as evidenced by Section 5.

3 APPROACH

3.1 Preliminary Study

The core idea of premise 1 is that, given a test input, if there is a large deviation in the prediction behavior between the compressed DNN model and the original model, it indicates that this test have a high probability to be misclassified by the compressed DNN model. In this section, to validate the rationality of Premise 1, we conducted the following preliminary study:

Objectives: We investigate the relationship between the prediction deviation resulting from model compression and the probability of the test being misclassified by the compressed DNN model.

Experimental design: We utilized a variety of distance measurement metrics, such as Euclidean Distance and Manhattan Distance, to investigate the correlation between prediction deviations and misclassification probabilities. For each distance metric, we evaluated the prediction deviations between the original DNN model and the compressed model for each sample in the test dataset, accordingly assigning a deviation score. Subsequently, we ranked all the test samples in descending order based on their deviation scores. We then divided the samples into ten equally sized groups, with deviations progressively decreasing across these ten segments. Within each segment, we identify the number of misclassified tests to observe whether there was a relationship between prediction deviation magnitude and the likelihood of misclassification.

Results: The experimental results of the preliminary study are presented in Table 1 and Figure 1. Table 1 displays the number of misclassified tests in different deviation levels of test groups based on various distance metrics. The "Deviated Behavior Metrics" in the table represent the metrics used to measure deviation. Furthermore, for each metric, we sorted all tests in the test set according to the magnitude of their deviation behavior. The range of 0%-10% indicates the top 10% of samples with the highest deviations. Similarly, 10% to 20% represents samples with deviation magnitudes falling within the top 10% to 20% interval. In each test group (i.e., 10%~20% and 20%~30%), the total number of samples is the same.

Table 1. Correlation between prediction deviation in the original model and the compressed model and misclassification of tests

Deviated Behavior Metrics	Percentage of tests selected									
	0%-10%	10%-20%	20%-30%	30%-40%	40%-50%	50%-60%	60%-70%	70%-80%	80%-90%	90%-100%
Euclidean	877	815	769	705	635	499	347	200	81	28
Manhattan	918	851	794	712	618	475	309	174	77	26
Chebyshev	868	811	755	690	628	504	356	230	86	27
SSD	877	815	769	705	635	499	347	200	81	28
Wasserstein	859	826	789	725	633	498	338	183	78	27

To provide a more precise illustration of the data in the table, we provide a concrete example: the number in the first row and first column of the table indicates that, when using the Euclidean method as the deviation metric, among the top 10% of tests with the highest deviation between the original model and the compressed model, there were 877 tests misclassified by the compressed model. It is important to note that the experiments for this research question were conducted using natural datasets, and the number of misclassified tests represents the mean across all 20 subjects.

We see that, as the deviation level decreases, regardless of the distance metric used to measure the deviation, the number of misclassified tests in each group decreases. For instance, with the Manhattan metric, in the top 10% of tests with the highest deviation, there were 918 misclassified tests. In the 10% to 20% deviation range, there were 851 misclassified tests, and this number decreased to 794 in the 20% to 30% range and further to 712 in the 30% to 40% range. Only 26 tests were misclassified in the 90% to 100% range. To visually represent this decreasing trend, we provide Figure 1, where each curve represents the relationship between deviation and misclassification for different distance metrics. In Figure 1, we see that as the deviation level decreases, the number of misclassified tests in the test groups gradually decreases. The above experimental results indicate that for a given test, if the original DNN model and the compressed model exhibit higher deviation in their predictions, the test is more likely to be misclassified by the compressed model.

Finding Tests with higher prediction disparities between the original DNN model and the compressed model are more likely to be misclassified by the compressed model.

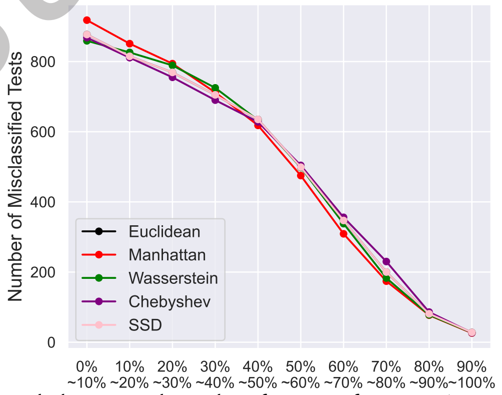

Fig. 1. Correlation between deviation behavior and misclassification of tests. X-Axis: Tests sorted by decreasing deviation; Y-Axis: the number of misclassified tests

3.2 Overview of PriCod

In this paper, we introduce PriCod, a novel test prioritization approach specifically designed for compressed Deep Neural Network (DNN) models. The overview of PriCod is depicted in Figure 2. Overall, the workflow of PriCod consists of four steps: Deviation Feature Generation, Embedding Feature Generation, Feature Fusion, and Feature-based Ranking. We provide a brief overview of these four steps below. For detailed explanations, please refer to Section 3.3 to Section 3.6.

❶ **Deviation Feature Generation** PriCod initially generates deviation features for each test input. These features capture the behavioral differences between the compressed DNN model and its original DNN model when predicting a given test input t. The utilization of deviation features for test prioritization is grounded in the premise that, for a given test, if the prediction behavior between the compressed DNN model and the original model exhibits a large deviation, this test is considered more likely to be misclassified by the compressed DNN model. (The preliminary study conducted in Section 3.1 further validates this premise).

❷ **Embedding Feature Generation** For each test case, PriCod also generates embedding features to reflect the key characteristics of the test (image/text). This step is based on the premise that test inputs closer to the model's decision boundary are more likely to be misclassified. By mapping each test input to a vector in space, the embedding features can indirectly indicate the proximity of the test to the decision boundary.

❸ **Feature Fusion** PriCod integrates deviation features and embedding features, generating a more comprehensive feature representation for each test input.

❹ **Feature-based Ranking** Utilizing the generated fused feature vector, PriCod employs the LightGBM model to calculate misclassification scores for each test input. A high score for a test implies that it is more likely to be misclassified by the compressed model. Therefore, PriCod ranks all tests in descending order based on the misclassification scores.

Below, we explain the rationale for the design.

- **Deviation feature generation - prioritizing tests based on behavioral deviation:** A key insight of PriCod is that if there is a high prediction deviation between the compressed model and the original model for a given test, it is highly likely that the compressed model has potential issues arising from the compression when handling this test. Therefore, PriCod utilizes the deviation information to perform test prioritization. The feasibility of this approach is demonstrated by the preliminary study conducted in Section 3.1.
- **Embedding feature generation - prioritizing tests based on proximity to the decision boundary:** An existing study [61] has pointed out that test inputs close to the decision boundary are more likely to be misclassified. PriCod generates embedding features to indirectly reflect the proximity of each test input to the decision boundary, thus performing test prioritization.
- **Feature fusion - enhancing the predictive power of the ranking model:** According to the existing study [68], feature fusion can enhance the predictive capabilities of models. Through feature fusion, we aim to enhance the ability of PriCod's ranking model to predict misclassification scores for each test, thereby improving the effectiveness of test prioritization.
- **Feature-based ranking - predicting the misclassification probability based on features:** Given a test input, PriCod utilizes the LightGBM model to estimate the probability of it being misclassified based on the fused features. LightGBM has been proven to be a powerful algorithm capable of predicting the probability values for different categories [44]. Within the framework of PriCod, LightGBM categorizes tests into two groups: those "misclassified by the compressed model" and those "not misclassified by the compressed model". We utilize LightGBM to predict the probability of each sample being misclassified by the compressed model, referred to as the misclassification score. A high misclassification score implies that the test is more likely to be misclassified by the compressed model. Therefore, PriCod ranks all tests in descending order based on the misclassification scores.

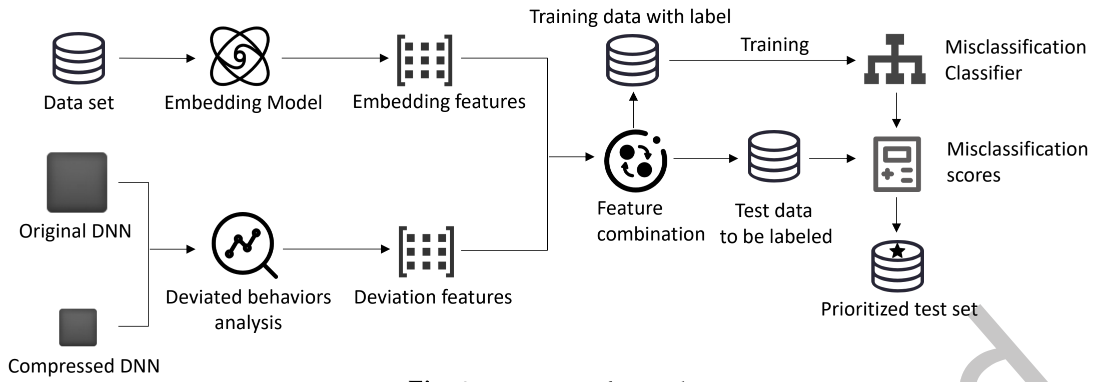

Fig. 2. Overview of PriCod

3.3 Deviation Features Generation

During the process of model compression, which aims at reducing storage and computational costs, the model can lose some details and complexity, resulting in a relatively simplified compressed model. Deviations in prediction behavior can arise between the original DNN model and the compressed DNN model. Our core premise for prioritizing testing based on these behavioral discrepancies is as follows: When there is a significant deviated behavior between the compressed DNN model and the original model for a given test, it suggests that the compressed model is more likely to produce a prediction different from that of the original model for this input. This test is deemed more likely to expose bugs in the compressed model. We validated premise 1 through a specially designed preliminary study. Further details can be found in Section 3.1.

In order to quantify the magnitude of differences in predictions between the original model and the compressed DNN model, we propose 17 strategies for generating deviation features, with the aim of effectively encapsulating variations in predictions between the original model and the compressed DNN model. Our objective is to provide a comprehensive suite of measures to capture different aspects of deviation before and after model compression for each test input.

- **Classification Deviation Features (CLA)** [93]: These features capture the disparities in predicted classes between the original DNN model and the compressed DNN model for a given test input. To derive these features, we compare the specific category predictions for each test case obtained from the original DNN model and the compressed DNN model. When the predictions differ, it signifies that the compressed model displays classification behavior contrary to that of the original model, providing a direct reflection of the variations in predictive behavior.
- **Confidence Deviation (CFD)** [29]: CFD reflects the absolute difference between the probability of the predicted category by the original model and the probability of the same category by the compressed DNN model. It reflects the degree of variation in the models' confidence levels.
- **Euclidean Distance Features (EUCL)** [56]: These features measure the Euclidean distance between the probability vectors of predictions made by the original model and the compressed model for a given test input. This distance metric reflects the overall magnitude of differences between the models' prediction probabilities, providing a comprehensive view of their predictive disparities.
- **Manhattan Distance Features (MHD)** [67]: MHD represents the Manhattan Distance between the probability vectors of predictions made by the original model and the compressed model for a given test input. It quantifies the sum of absolute differences between corresponding probabilities, indicating the overall shift in predictions and the directions of these shifts.

- **Chebyshev Distance Features (CHD)** [47]: CHD reflects the Chebyshev Distance between the probability vectors of predictions made by the original model and the compressed model for a specific test input. This metric highlights the maximum absolute difference between corresponding probabilities, showcasing the most significant deviations in the models' predictions.
- **Pearson Correlation Coefficient Features (PCC)** [16]: PCC measures the Pearson Correlation Coefficient between the probability vectors of predictions made by the original model and the compressed model for a given test input. It indicates the strength and direction of a linear relationship between the predictions, offering insights into the consistency of their deviations. A higher PCC value indicates smaller disparities in predictive behavior between the original model and the compressed model, while a lower PCC value indicates relatively larger disparities.
- **Sum of Squared Differences Features (SSD)** [81]: SSD reflects the Sum of Squared Differences between the probability vectors of predictions made by the original model and the compressed model for a specific test input. It emphasizes larger deviations while downplaying smaller ones, providing a measure of the overall prediction divergence.
- **Hellinger Distance Features (HED)** [31]: HED reflects the Hellinger Distance between the probability vectors of predictions made by the original model and the compressed model for a given test input. It measures the similarity between the square root of the prediction probabilities, offering insights into their predictive differences.
- **Wasserstein Distance Features (WAS)** [75]: WAS reflects the Wasserstein Distance between the probability vectors of predictions made by the original model and the compressed model for a specific test input. It measures the minimum "cost" of transforming one distribution into another, highlighting their divergence.
- **Coordinate Deviation Features (CDF)**: CDF is obtained by subtracting the origin coordinates from the prediction vector of the compressed model. This vector directly reflects the deviation in predictions of the compressed DNN models from the origin coordinates.
- **Relative Entropy Features (REL)** [8]: REL reflects the Relative Entropy between the probability vectors of predictions made by the original model and the compressed model for a given test input. It measures the information lost between the compressed model and the original model.
- **Difference Vector Features (DIF)**: DIF refers to the vector obtained by subtracting the prediction vector of the original model from the prediction vector of the compressed model for a specific test input. This vector directly reflects the direction and magnitude of deviation in predictions.

The aforementioned deviation features exhibit the following differences:
1) Classification Deviation Features (CLA) vs. Confidence Deviation (CFD) CLA focuses on the difference in classification results, i.e., whether the compressed model produces the same output classification for a given test as its original model. CFD measures the deviation in confidence, indicating the differences in prediction uncertainty between a compressed model and its original model for a given test.
2) Euclidean Distance (EUCL) vs. Manhattan Distance (MHD) EUCL measures the straight-line distance between the prediction vectors of the compressed DNN model and its original DNN model, making it suitable for quantifying overall deviations in predictions in a multi-dimensional space. On the other hand, MHD calculates the sum of absolute differences in each dimension based on the prediction vectors.
3) Chebyshev Distance (CHD) vs. Sum of Squared Differences (SSD) CHD emphasizes the difference in the maximum single dimension of the prediction vectors, particularly highlighting extreme values. SSD represents the sum of squared differences across dimensions, offering a quantitative measure of overall deviation.
4) Hellinger Distance (HED) vs. Wasserstein Distance (WAS) HED measures the disparities in the shape of distributions between the prediction vectors generated by the compressed model and those of its original model for a specific test. In contrast, WAS focuses on the "effort" required to transform one distribution into another.

5) Relative Entropy (REL) vs. Pearson Correlation Coefficient (PCC) REL specifically focuses on information loss or gain to quantify the disparity in prediction distributions between the compressed model and its original model. On the other hand, PCC measures the degree of linear correlation among the prediction vectors.
6) Coordinate Deviation Features (CDF) vs. Difference Vector Features (DIF) Given a test input, CDF reflects the absolute prediction bias between predictions made by the compressed model and those by the original model. On the other hand, DIF emphasizes changes in both direction and magnitude relative to the predictions of the original model and the compressed model.

The aforementioned deviation features have proven to be useful in the context of PriCod for test prioritization, as evidenced by the preliminary study conducted in Section 3.1 and RQ5 (Feature contribution analysis). Specifically, the rationale behind these features is derived from PriCod's premise 1, which states that for a given test, if the prediction behavior between the compressed DNN model and the original model exhibits a large deviation, it suggests that this test is more likely to be misclassified by the compressed DNN model. The aforementioned features are all designed to measure the prediction differences. Therefore, these features can be utilized for test prioritization. The validation of premise 1's reasonability is established in the preliminary study (cf. Section 3.1), and the effectiveness of deviation features is further demonstrated in RQ5 (cf. Section 5.5), which leverage ablation studies to confirm the contributions of the deviation features.

3.4 Embedding Features Generation

Embedding Features (EF) capture the intrinsic information of each test input $t \in T$. In our experiments, PriCod was evaluated under two scenarios: image-type tests and text-type tests. In the following sections, we present the generation process of embedding features under each of the scenarios. In the context of image-type tests, to obtain these Embedding Features (EFs), we utilize a pre-trained ResNet model [34] to transform each test into a vector representation. In the case of text-type tests, we employ the BERT [21] model to map each input into a corresponding vector representation.

The process by which ResNet transforms an image into an embedding feature vector is as follows: First, the pre-trained ResNet network is employed to load the image. Through successive layers of convolution and pooling, the image undergoes a gradual transformation into higher-level abstract features. The output of the final global average pooling layer is extracted and treated as the image's embedding. This embedding encapsulates the semantic information of the image.

The principle behind prioritizing testing using the Embedding Features of test cases is that: *test inputs closer to the decision boundary of the model are more likely to be misclassified*, as outlined in existing literature [61]. By mapping images to vectors in space, PriCod can automatically learn the distance between a test and the decision boundary to perform effective test prioritization. The benefits of utilizing the ResNet model to generate embedding features are outlined below:

- **Automatic Identification of Vital Features** The ResNet model can automatically extract crucial features from raw data. By generating embedded features, key characteristics can be focused.
- **Effective Feature Generation** ResNet boasts powerful feature generation capabilities. Its architecture integrates multiple convolutional and pooling layers, allowing the model to effectively capture a wide range of features and patterns within images. This capability can be highly advantageous for test prioritization.

The process by which BERT converts textual data into an embedding feature vector can be described in the following steps:

- **Tokenization** The input text, whether a sentence or a paragraph, is broken down into smaller units called tokens. These tokens are typically words or subwords, and each is assigned a unique identifier.
- **Embedding** Each token identifier is transformed into a corresponding word vector. These vectors are rich in semantic information, representing not just the token but also aspects of its meaning.

- **Contextual Analysis with Transformer Encoders** BERT utilizes multiple layers of Transformer encoders to process these word vectors. These encoders are adept at capturing contextual information, allowing the model to understand the nuances and varied meanings of words based on their context in the sentence or paragraph.
- **Generation of Embedding Vectors** The final hidden states outputted by the Transformer encoders serve as the embedding vectors. These vectors represent the entire input text (sentence or paragraph) and encapsulate both the semantic and contextual information of the original text.

The advantages of the BERT model include: BERT takes into account contextual information when processing text, and the vectors it generates can better represent the semantic meaning of the text. As a result, when text information is mapped into space, BERT can place semantically similar texts closer together in this space, with texts of the same category being nearer to each other and different texts being farther apart. These vectors in space can indirectly reflect the distance from the decision boundary.

3.5 Feature Fusion

Building upon the above procedures, PriCod produces two distinct categories of feature vectors for each test sample in T: deviation feature vector and embedding feature vector. Following this, for each test sample $t \in T$, PriCod combines the two feature vectors and input to the LightGBM classifier, facilitating the prediction of the misclassification score for the test case. The process of feature fusion serves to enhance the effectiveness of subsequent test prioritization. Each type of feature (deviation features, embedding features) captures distinct aspects of the data, which hold informative value for the final prioritization. Through the amalgamation of these features into a singular feature vector, multiple sources of information are effectively integrated.

3.6 Feature-based Ranking

After obtaining the feature vector for each test t in set T, PriCod employs the **LightGBM classifier** [44] as the ranking model to predict the misclassification probability for each t based on its corresponding feature vector. LightGBM is a gradient-boosting framework that employs decision trees as base learners. LightGBM is designed for efficient parallel computation. In the subsequent section, we elaborate on the procedures of constructing the classifier and elucidate the adaptations carried out on the classifier to generate the misclassification scores instead of producing categories.

- **Construction of the LightGBM Classifier** Given the compressed DL model M and the evaluated dataset, to build the classifier, we first partitioned the dataset into two sets: a training set labeled as R and a test set labeled as T, with a partition ratio of 7:3 [72]. The test set T remains untouched for evaluating PriCod. The LightGBM ranking model is trained using the dataset R', which is derived from the original training set R of the evaluated compressed model. The process of constructing the training set for the ranking model R' is described below. Initially, we generate deviation features and embedding features for each instance $r \in R$. These features serve as the training features for the new dataset R'. Subsequently, we construct the labels for R'. To this end, we employ the compressed DNN model M to predict the classification of each instance $r \in R$. We compare the model's predictions with the corresponding ground truth of r to determine whether r is misclassified. Instances that are misclassified are labeled as 1, while correctly classified instances are labeled as 0. Consequently, we obtain the labels for the training set of the ranking model. By utilizing the constructed training set, we train the ranking model LightGBM.
- **Adapting the LightGBM Classifier** Once the training is complete, LightGBM can be used to predict whether a test input will be misclassified by the compressed DNN model. To enable the model to produce misclassification scores as outputs instead of labels, we introduced specific modifications to the original LightGBM classifier. More precisely, we extract the intermediate value from the model's output, which initially served to determine

whether a test would be misclassified by the model. In the model's decision-making process, if this intermediate value exceeds a certain threshold, the input is classified as "misclassified"; otherwise, it is classified as "not misclassified." Within the adjusted LightGBM classifier, this intermediate value refers to the misclassification score. A higher value signifies that a test instance is more likely to be misclassified.

- **Test Prioritization** Ultimately, PriCod ranks all the tests within the test set T in a descending order based on their respective misclassification probability scores. This sorting procedure yields the prioritized test set denoted as T'.

In the above steps of constructing the LightGBM classifier, we utilized deviation features and embedding features as training features to train the LightGBM model. We selected these specific features due to their ability to reflect the probability of a test being misclassified by a compressed DNN model. The rationale behind selecting these features is grounded in the core premises of PriCod (cf. Section 3.3 and Section 3.4):

- **Premise 1 (for deviation features)** For a given test, if the prediction behavior between the compressed DNN model and the original model exhibits a large deviation, it suggests that this test is more likely to be misclassified by the compressed DNN model. This premise indicates that deviation features can reflect the probability of a test being misclassified. The preliminary study conducted in Section 3.1 further validates this premise.
- **Premise 2 (for embedding features)** Test inputs that are located closer to the decision boundary of the model are more likely to be misclassified [61]. This premise elucidates that embedding features can reflect the probability of a test being misclassified. Specifically, by mapping each test input to a vector in space, the embedding features can indirectly indicate the proximity of the test to the decision boundary.

In the training set, each training sample is labeled as 0 or 1. Specifically, 0 means the sample was not misclassified by the compressed DNN model under evaluation, while 1 indicates the sample was misclassified. PriCod first generates deviation features and embedding features from each training sample. Using the LightGBM model, it learns the relationship between these features and "being misclassified". Following the completion of training, given a new test input, PriCod can quantify the probability of this test being misclassified based on its deviation features and embedding features.

3.7 Variants of PriCod

To comprehensively investigate how different feature fusion methods affect the overall performance of PriCod, we propose three distinct PriCod variants. These variants were meticulously designed to incorporate different approaches for combining embedding features and deviation features, shedding light on the impact of feature fusion techniques on PriCod's effectiveness. These three variants are denoted as PriCod^a, PriCod^m, and PriCod^c, each adopting a unique strategy for feature fusion: addition, multiplication, and cross-multiplication [59], respectively.

- **PriCod^a** The variant PriCod^a utilizes an addition-based fusion method for test prioritization. In this variant, the feature fusion strategy relies on addition, combining the embedding features and deviation features by adding them. Addition-based fusion is straightforward and results in a merged feature vector where the corresponding elements of the two input feature vectors are summed together.
- **PriCod^m** The variant PriCod^m employs a multiplication-based fusion method for test prioritization. This approach involves multiplying each element of the embedding features by the corresponding element of the deviation features. Multiplication-based fusion is more intricate compared to addition-based fusion and has the potential to emphasize interactions and relationships between the two feature sets.
- **PriCod^c** PriCod^c employs a cross-multiplication feature fusion strategy. This approach performs cross-multiplication on elements of the embedding features and deviation features. Cross-multiplication is a more complex fusion method compared to both addition and multiplication. It allows the model to capture intricate interdependencies between features by considering all possible pairwise interactions.

To enhance the clarity of each feature fusion strategy, we incorporate an illustrative example. Given a test t, we first follow the steps outlined in Section 3.3 and Section 3.4 to obtain its embedding feature vector and deviation feature vector. Assuming these two vectors are $\{e1, e2, e3\}$ (embedding features) and $\{d1, d2, d3\}$ (deviation features), we present the final vectors obtained after addition-based fusion, multiplication-based fusion, and cross-multiplication-based fusion below. **Addition-based fusion:** $\{e1 + d1, e2 + d2, e3 + d3\}$. **Multiplication-based fusion:** $\{e1 \times d1, e2 \times d2, e3 \times d3\}$. **Cross-multiplication-based fusion:** $\{e1 \times d1, e1 \times d2, e1 \times d3, e2 \times d1, e2 \times d2, e2 \times d3, e3 \times d1, e3 \times d2, e3 \times e3\}$. In the following, we explain the rationale behind the studied feature fusion strategies.

- **Feature fusion based on addition (PriCod^a)** The rationale behind this fusion approach mainly consists of three points. 1) This approach is intuitive, simply adding the information of two feature sets together, making it easy to understand and implement. 2) by simple addition, this method preserves the complete information of each feature set. 3) This approach is widely used in the context of DNNs and has been proven to be effective [34].
- **Feature fusion based on multiplication (PriCod^m)** The rationale behind the multiplication-based fusion approach mainly consists of three points. 1) Multiplication-based fusion emphasizes the interdependence between features. 2) Multiplying certain input elements by smaller weights can contribute to ignoring irrelevant information.
- **Feature fusion based on cross-multiplication (PriCod^c)** 1) The cross-multiplication fusion allows the model to capture more complex interdependencies and interactions between features by considering possible pairwise interactions. 2) This approach creates a high-dimensional feature space capable of revealing more hidden patterns and relationships. 3) Existing research [59] has proven the effectiveness of this method.

While we have introduced diversity in the feature fusion aspect, we intentionally maintained all other aspects of the PriCod variants identical to the original PriCod, aiming to ensure that any observed performance differences can be attributed primarily to the selected fusion method.

4 STUDY DESIGN

In this section, we present a comprehensive elucidation of the specific details concerning our study design. To begin, Section 4.1 elucidates the research questions that guided our investigation. Subsequently, Section 4.2 provides intricate insights into the compressed models and datasets adopted in our study. Section 4.3 showcases the noisy generation techniques utilized in RQ2, while Section 4.4 exhibits the adversarial attacks employed in the context of RQ3. Moreover, Section 4.5 demonstrates the test prioritization methods subjected to comparison. In addition, Section 4.6 outlines the measurement metrics we employed to assess the effectiveness of PriCod, its variants, and the compared approaches. Finally, Section 4.7 provides a comprehensive overview of the implementation and configuration setup utilized throughout our study.

4.1 Research Questions

Our experimental evaluation answers the research questions below.

- **RQ1: How does PriCod perform in prioritizing test inputs for compressed DNN models?**
  When utilizing confidence-based test prioritization techniques for compressed DNN models, these methods typically treat the compressed DNN models as black boxes, neglecting valuable information about deviations before and after model compression in the prioritization process. In our research, we introduce PriCod, a tailored test prioritization approach explicitly designed for compressed DNNs. PriCod harnesses prediction disparities induced by model compression, in combination with test input embeddings, to efficiently prioritize tests that may reveal misclassifications. In this study, we assess the effectiveness of PriCod by comparing it to a set of existing test prioritization methods.
- **RQ2: How does PriCod perform on noisy test inputs?**

Table 2. Compressed DNN models and datasets

ID	Dataset	# Size	Model	Compression Tool	Supported Mobile Platforms
1	CIFAR10	60,000	AlexNet-coreml	Core ML	iOS, watchOS
2	CIFAR10	60,000	AlexNet-tflite	TensorFlow Lite	Android, Linux-based Systems
3	CIFAR10	60,000	VGG16-coreml	Core ML	iOS, watchOS
4	CIFAR10	60,000	VGG16-tflite	TensorFlow Lite	Android, Linux-based Systems
5	Fashion	70,000	LeNet1-coreml	Core ML	iOS, watchOS
6	Fashion	70,000	LeNet1-tflite	TensorFlow Lite	Android, Linux-based Systems
7	Fashion	70,000	LeNet5-coreml	Core ML	iOS, watchOS
8	Fashion	70,000	LeNet5-tflite	TensorFlow Lite	Android, Linux-based Systems
9	Plant	52,803	NIN-coreml	Core ML	iOS, watchOS
10	Plant	52,803	NIN-tflite	TensorFlow Lite	Android, Linux-based Systems
11	Plant	52,803	VGG19-coreml	Core ML	iOS, watchOS
12	Plant	52,803	VGG19-tflite	TensorFlow Lite	Android, Linux-based Systems
13	CIFAR100	60,000	ReseNet152-coreml	Core ML	iOS, watchOS
14	CIFAR100	60,000	ReseNet152-tflite	TensorFlow Lite	Android, Linux-based Systems
15	CIFAR100	60,000	DenseNet201-coreml	Core ML	iOS, watchOS
16	CIFAR100	60,000	DenseNet201-tflite	TensorFlow Lite	Android, Linux-based Systems
17	News	21,107	LSTM-coreml	Core ML	iOS, watchOS
18	News	21,107	LSTM-tflite	TensorFlow Lite	Android, Linux-based Systems
19	News	21,107	GRU-coreml	Core ML	iOS, watchOS
20	News	21,107	GRU-tflite	TensorFlow Lite	Android, Linux-based Systems

When implemented on mobile devices, compressed DNN models can encounter noisy data due to a range of factors, such as capturing photos from various angles, as well as the presence of raindrops. In this research inquiry, we employ a set of noise generation techniques [69, 80, 85, 89] to construct noisy datasets for evaluating the performance of PriCod.

- **RQ3: How does PriCod perform on adversarial inputs?**
  The previous research questions have assessed PriCod's effectiveness with natural and noisy test inputs. In this research question, we evaluate PriCod's performance with adversarial test inputs.
- **RQ4: How does the feature combination strategies impact the effectiveness of PriCod?**
  In this research question, we aim to gain a deeper understanding of the impact of different feature combination strategies on the effectiveness of PriCod. By analyzing this critical factor, we can determine which feature fusion technique is better suited for PriCod.
- **RQ5: To what extent does each type of features contribute to the effectiveness of PriCod?**
  In this research question, we investigate the impact of various feature types on the performance of PriCod. This investigation enables us to identify the features that have a more significant influence on PriCod. Additionally, conducting a thorough analysis of feature contributions can enhance our comprehension of the underlying mechanisms and operational principles of PriCod.
- **RQ6: To what extent can uncertainty-based metrics contribute to improving the effectiveness of PriCod?**
  In the test prioritization process within PriCod, we generate embedding features for each test to indirectly reveal the proximity between the test and the decision boundary. A prior study [99] indicated that uncertainty-based metrics can also reflect the proximity. Therefore, in this research question, we investigate whether incorporating uncertainty-based metrics can enhance the effectiveness of PriCod. To be more specific, we employ several uncertainty metrics (such as DeepGini [29] and Margin [96]) to produce uncertainty features for each test and integrate them into the original PriCod for test prioritization.

4.2 Models and Datasets

In our study, we have utilized a total of 182 subjects to assess the performance of PriCod and the compared approaches [29, 39]. Table 2 provides essential particulars regarding these subjects, encompassing the dataset-model associations, dataset sizes, tools used for DNN compression, and supported mobile platforms. Among the 182 subjects under investigation in this study, 20 subjects are constructed based on natural datasets, 126 subjects are built on noisy datasets, and 36 subjects are established on adversarial datasets.

4.2.1 Datasets.

In our study, we evaluate the performance of PriCod using five distinct datasets: CIFAR10 [48], Fashion [102], Plant [70], CIFAR100 [84], and News [110]. The selection of these specific datasets is grounded in their widespread adoption within the domain of DNN testing. Notably, CIFAR10 and Fashion are two of the most widely employed datasets for DNN evaluation. The Plant dataset stands out as a renowned dataset for AI applications focused on detecting plant diseases. Moreover, compressed models tailored to this context have already been implemented. Therefore, investigating this dataset becomes particularly valuable for the study of compressed models.

- **CIFAR10** [48] The CIFAR10 dataset serves as a frequently utilized collection of images for training and assessing DNN models. Comprising a total of 60,000 images, each measuring 32x32 pixels and in color, the dataset is categorized into ten distinct classes, containing 6,000 images per class. These categories include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- **Fashion** [102] The Fashion dataset comprises fashion product images from 10 categories, such as T-shirts, trousers, dresses, etc. Each category consists of 7000 images, making a total of 70,000 images. All images in the dataset are grayscale with dimensions of 28x28 pixels.
- **Plant** [70] The Plant dataset covers a diverse range of crops and various disease types, including images depicting healthy plant leaves. Specifically, the Plant dataset comprises 52,803 plant leaf images organized into 38 distinct category labels. Each label corresponds to a combination of a specific crop and a disease. Examples of these classes encompass Healthy Corn Leaf, Potato Late Blight, and Rose Black Spot.
- **CIFAR100** [84] The CIFAR100 dataset is a widely used benchmark in computer vision, consisting of 60,000 32x32 color images categorized into 100 distinct classes (such as clouds, cups, and forests). CIFAR100 is a valuable resource for training and evaluating DNN models, particularly designed for image classification tasks.
- **News** [110] The News dataset is an English-language dataset that comprises an annotated corpus of finance-related tweets. This dataset is utilized for the classification of finance-related tweets based on their topics. It consists of 21,107 samples categorized into 20 classes. Examples of these categories include financials, currencies, and opinion.

4.2.2 Compressed DNN models.

Our study encompasses a set of 20 compressed deep neural network (DNN) models. Our approach for model compression primarily focuses on model quantization, a prevalent method in the field of model compression. The reason for selecting quantization as our compression method is that, in the industry, quantization is considered one of the most commonly adopted techniques for deploying models to mobile devices [38, 53]. In the process of generating compressed models, we primarily employed two quantization techniques to compress DNN models. Below, we provide details regarding the two techniques used to compress DNN models:

- **Tensorflow Lite** [19] We utilized TensorFlow Lite (TFLite) as one of the compression techniques to compress DNN models. TFLite, a component of the TensorFlow deep learning framework developed and maintained by Google, offers interfaces for converting TensorFlow models into lightweight counterparts. This facilitates deployment on various low-computing devices, including Android mobile phones. In our experiments, we chose 8-bit quantization for model compression. This involves quantizing the weights and activation values

in the model to 8-bit numbers, thereby reducing the model's size and enhancing deployment efficiency on resource-constrained devices.
- **CoreML** [90] In our experiments, we utilized another pivotal technique for model compression, namely CoreML. Developed by Apple, CoreML is a framework designed to transform models into the Mlmodel format, customized for iOS platforms. Similar to parameter selection in TFLite, we implemented 8-bit quantization. This process involves quantizing the model's weights and activation values into 8-bit numbers.

In the following, we present detailed information on all the compressed DNN models employed to evaluate PriCod.

- **AlexNet-coreml and AlexNet-tflite** AlexNet [49] is a deep convolutional neural network designed for image classification tasks. It comprises multiple convolutional layers, pooling layers, and fully connected layers, all using the Rectified Linear Unit (ReLU) activation function. ***AlexNet-coreml*** is a compressed version of the AlexNet model converted into the CoreML format, suitable for inference and applications on Apple devices. ***AlexNet-tflite*** is a compressed version of the AlexNet model converted into the TensorFlow Lite (TFLite) format, designed for inference and applications on Android and embedded devices.
- **VGG16-coreml and VGG16-tflite** VGG16 [86] is a deep convolutional neural network with 16 convolutional and fully connected layers, primarily used for image classification. Its notable features include fixed 3x3 convolutional kernel size and a large number of layers, making it suitable for training on large-scale image datasets. ***VGG16-coreml*** is a compressed version of the VGG16 model converted into the CoreML format intended for image processing tasks on Apple devices. ***VGG16-tflite*** is a compressed version of the VGG16 model converted into the TFLite format, designed for image processing tasks on Android and embedded devices.
- **LeNet1-coreml and LeNet1-tflite** LeNet-1 [51] is an early convolutional neural network. It consists of convolutional layers, pooling layers, and fully connected layers, suitable for small-scale image classification tasks. ***LeNet1-coreml*** is a compressed version of the LeNet-1 model converted into the CoreML format intended for image processing and recognition tasks on Apple devices. ***LeNet1-tflite*** is a compressed version of the LeNet-1 model converted into the TFLite format.
- **LeNet5-coreml and LeNet5-tflite** LeNet-5 [51] is another early convolutional neural network. It includes convolutional layers, pooling layers, and fully connected layers and has more layers and parameters compared to LeNet-1. ***LeNet5-coreml*** is a compressed version of the LeNet-5 model converted into the CoreML format. ***LeNet5-tflite*** is a compressed version of the LeNet-5 model converted into the TFLite format.
- **NIN-coreml and NIN-tflite** NIN [57] is an innovative convolutional neural network architecture that introduces the concept of "network in network" to enhance model expressiveness. It uses 1x1 convolutional layers to extract local features. ***NIN-coreml*** is a compressed version of the NIN model converted into the CoreML format, suitable for image processing tasks on Apple devices. ***NIN-tflite*** is a compressed version of the NIN model converted into the TFLite format.
- **VGG19-coreml and VGG19-tflite** VGG19 [86] is an extended version of VGG16 with more convolutional and fully connected layers, suitable for complex image classification tasks. ***VGG19-coreml*** is a compressed version of the VGG19 model converted into the CoreML format. ***VGG19-tflite*** is a compressed version of the VGG19 model converted into the TFLite format.
- **ReseNet152-coreml and ReseNet152-tflite** ResNet152 [34] is a deep convolutional neural network (CNN) employing the Residual Network architecture, with 152 layers and 60.4 million parameters. ***ResNet152-coreml*** represents a compressed version of the ResNet152 model, converted into the CoreML format. Similarly, ***ResNet152-tflite*** is a compressed variant of the ResNet152 model, converted into the TFLite format.
- **DenseNet201-coreml and DenseNet201-tflite** DenseNet201 [40] is a convolutional neural network consisting of 201 layers and a total of 20.2 million parameters. ***DenseNet201-coreml*** denotes a compressed version

of the DenseNet201 model, converted into the CoreML format. ***DenseNet201-tflite*** is another compressed version of the DenseNet201 model, converted into the TFLite format.

- **LSTM-coreml and LSTM-tflite** [108] LSTM (Long Short-Term Memory) is a type of recurrent neural network known for its ability to capture long-term dependencies in sequential data. ***LSTM-coreml*** refers to a compressed version of the LSTM model, converted into the CoreML format. ***LSTM-tflite*** is another compressed version of the LSTM model, converted into the TFLite format.
- **GRU-coreml and GRU-tflite** [22] GRU (Gated Recurrent Unit) is a recurrent neural network architecture widely employed for processing sequential data. Compared to LSTM, GRU is characterized by fewer gates and parameters, making it faster. ***GRU-coreml*** denotes the compressed version of the GRU model, converted into the CoreML format. ***GRU-tflite*** is another compressed version of the GRU model, converted into the TFLite format.

4.3 Noise Generation Techniques

In our experiments for Research Question 2 (RQ2), we utilized 13 noise techniques sourced from top-level conferences [69, 80, 85, 89]. These diverse selections of noise generation techniques were aimed at evaluating the effectiveness of PriCod in a broader range of noisy scenarios. We provide a detailed explanation of each noise-generation technique below.

- **Channel Shift Range (CSR)** The CSR technique engenders a transformative alteration in the image's overall color palette by perturbing the values of its color channels.
- **Feature-wise Std Normalization (FSN)** FSN operates by normalizing each input sample with respect to its standard deviation. The underlying motivation is to decentralize the dataset.
- **Height Shift (HS)** HS effectuates vertical displacements of an image, essentially shifting it upwards or downwards within the image canvas. This spatial modification introduces variations in the vertical positioning of objects.
- **Horizontal Flip (HF)** The HF technique orchestrates horizontal mirroring of the input image. By introducing random horizontal flips during augmentation, diverse perspectives of objects are captured.
- **Vertical Flip (VF)** VF introduces a vertical inversion of the image, essentially flipping it along the horizontal axis.
- **Rotation (RO)** RO introduces controlled rotations to the input samples, adhering to a designated angle range.
- **Shear Range (SR)** SR engenders a shear transformation, preserving one coordinate while linearly displacing the other.
- **Width Shift (WS)** WS pertains specifically to horizontal translations, thereby repositioning the image horizontally within the canvas.
- **Zca Whitening (ZCA)** ZCA performs dimensionality reduction on the input images, effectively reducing redundancy while retaining essential features.
- **Zoom (ZOO)** ZOO introduces alterations in the image scale by magnifying or contracting it along its length or width.
- **Contrast (CON)** CON quantifies the disparity between the brightest and darkest regions of an image. This augmentation manipulates image contrast, diversifying the dataset concerning luminance and accentuating differences between light and dark areas.
- **Noise Gasuss (GAS)** GAS simulates signal noise by following a Gaussian distribution. By adding this form of noise to the input images, the dataset's resilience to stochastic variations is bolstered.
- **Salt & Pepper (SP)** SP augments the dataset by introducing either white or black pixels to the image, imitating the effects of salt and pepper noise. This introduces localized distortions.

4.4 Adversarial Techniques

In the context of RQ3 in our study, we employed four distinct adversarial techniques to generate adversarial samples for assessing the effectiveness of PriCod. These techniques include the Fast Gradient Method, Adversarial Patch, Basic Iterative Method and Projected Gradient Descent. We elaborate on the operational principles of each adversarial technique in the following explanations.

- **Fast Gradient Method (FGM)** [32] FGM is an extension of the Fast Gradient Sign Method (FGSM), which was the pioneering gradient-based white-box attack algorithm utilizing deep neural network gradients to craft adversarial examples. FGM enhances the original FGSM attack by incorporating other norms for perturbation generation.
- **Adversarial Patch (Patch)** [5] AP generates adversarial examples by creating attack patches designed to replace specific portions of the original images. Importantly, this technique does not necessitate attackers to possess knowledge about the original dataset.
- **Basic Iterative Method (BIM)** [50] BIM is an advancement of FGSM involving multiple iterations of small perturbations. After each iteration, the pixel values of the obtained result are clipped to ensure that the outcome remains within the vicinity of the original image.
- **Projected Gradient Descent (PGD)** [63] PGD attack is an iterative technique. In contrast to FGSM, which involves a single iteration and a significant perturbation, PGD incorporates multiple iterations with small perturbations. During each iteration, the perturbation is constrained within predefined boundaries.

4.5 Compared Approaches

To demonstrate the effectiveness of PriCod, we employed seven test prioritization approaches along with a baseline method. These seven methods are DeepGini, Prediction-Confidence Score (PCS), Vanilla Softmax, Entropy, Margin, Least Confidence (LC) and ATS. We chose these methods for the following reasons: 1) These approaches can be tailored to prioritize tests for compressed Deep Neural Networks; 2) These approaches have previously shown effectiveness for DNNs; 3) These approaches offer open-source implementations. It is crucial to emphasize that all the test prioritization methods used for comparison were initially designed for non-compressed models in their respective research papers. However, despite being designed for uncompressed models, these methods can be directly applied to compressed DNN models. This is a crucial factor why we selected them to compare with PriCod.

- **DeepGini** [29] DeepGini performs test prioritization by calculating the model confidence towards each test case. The metric used to measure the confidence score is the Gini coefficient, which is calculated using the Formula 1 provided below. A higher Gini coefficient for a test indicates that the test is more likely to be misclassified. Therefore, it should be prioritized towards the front of the test set.

$$Gini(t) = 1 - \sum_{i=1}^{N} (p_i(t))^2 \tag{1}$$

  where $Gini(t)$ represents the Gini score of the test t. $p_i(t)$ denotes the probability that the test input t is predicted to belong to label i. N represents the total number of classes to which the input can be classified.
- **Prediction-Confidence Score (PCS)** [99] For each test input in the specified test set, PCS calculates the difference between the probability of the model's most confident prediction for that test and the probability of the second most confident prediction. A smaller difference indicates that the model is less confident in its prediction for that particular test input, and this input will be prioritized higher. The formula for this calculation is provided in Formula 2.

$$PCS(t) = p^1(t) - p^2(t) \tag{2}$$

where $p^1(t)$ denotes the probability of the model's most confident prediction for that test, and $p^2(t)$ represents the probability of the model's second most confident prediction for the same test.

- **Vanilla Softmax** [99] Vanilla Softmax prioritizes tests by calculating the difference between the maximum activation probability in the output softmax layer and the ideal value of 1 for each test input. Test inputs exhibiting larger disparities are perceived as more likely to be misclassified by the model. The computation of Vanilla Softmax is demonstrated in Formula 3.

$$\mathrm{V}(t) = 1 - \max_{i=1}^{N} p_i(t) \tag{3}$$

where $\max_{i=1}^{N} p_i(t)$ represents the maximum activation probability in the output softmax layer for the test input t. Here, N denotes the total number of prediction classes. $p_i(t)$ signifies the probability that the model classify the test t into class i.

- **Entropy** [99] Entropy prioritizes tests by computing the entropy of the softmax likelihood for each test instance. Higher entropy values imply higher uncertainty in the model's predictions for those inputs. Therefore, test inputs with greater entropy are interpreted as more prone to being misclassified by the model and will be assigned higher priority.
- **Margin** [96] Margin prioritizes tests by evaluating the difference between the model's most confident prediction and the second most confident prediction for each test. For a given test, if its margin score is large, the test is considered more likely to be misclassified. The margin score is calculated by Formula 4.

$$M(t) = p_k(t) - p_j(t) \tag{4}$$

where $M(x)$ denotes the margin score. $p_k(t)$ represents the model's most confident prediction probability for the test instance t. $p_j(t)$ represents the second most confident prediction probability.

- **Least Confidence (LC)** [96] Least Confidence regards test inputs for which the model exhibits the least confidence as more likely to be misclassified. The least confidence score is calculated using Formula 5. For a given test, a higher least confidence score indicates that the model is less confident about the prediction for that particular test. Therefore, this test is considered more likely to be predicted incorrectly.

$$L(t) = 1 - \max_{i=1:n} p_i(t) \tag{5}$$

where $L(t)$ represents the least confidence score. $p_i(t)$ denotes the probability that the test input t is predicted to be label i via a model M. $\max_{i=1:n} p_i(t)$ corresponds to the model's most confident prediction probability for the test instance t.

- **ATS** [30] ATS (Adaptive Test Selection) is an adaptive method for test selection that utilizes variations in model outputs to assess the behavioral diversity of DNN test data. Its objective is to select a diverse subset of tests from a massive unlabeled dataset. In empirical evaluations [30], ATS has demonstrated superior performance compared to all the evaluated coverage-based test selection methods, showing significant improvements in both fault detection and model improvement capabilities.
- **Random selection** [25] Random selection serves as the baseline in our study. This approach involves randomly determining the order in which test inputs are executed. This implies that the arrangement of test inputs is established entirely at random, without any predefined patterns or logical sequences.

In addition to the aforementioned test prioritization methods, there are several classic approaches in the literature for prioritizing tests for DNNs, including PRIMA [97] and coverage-based test prioritization methods [79]. However, due to the characteristics of compressed models, these methods are challenging to directly apply. We explain the specific reasons below:

- **PRIMA** PRIMA is not suitable for compressed DNN models primarily because some of its model mutation rules cannot be adapted to the structure of the compressed DNN model. For example, one of PRIMA's model mutation operations is Neuron Activation Inverse, which reverses the activation state of a neuron by changing the sign of the neuron output before passing it to the activation function. However, previous research [93] has pointed out that the structure of compressed models does not support such model mutation operations.

- **Coverage-based metrics** Coverage-based test prioritization methods, such as DeepXplore, cannot be applied to compressed DNN models primarily due to the fact that coverage-based approaches typically prioritize test inputs based on their neuron coverage. However, existing study [93] pointed out that, due to the unique structure of compressed models, obtaining neuron coverage is not feasible. Therefore, coverage-based test prioritization methods cannot be adapted to compressed DNN models.

4.6 Measurements

Consistent with previous studies [29], we utilized two metrics to evaluate the effectiveness of PriCod: Average Percentage of Fault Detection (APFD) [106] and Percentage of Faults Detected (PFD) [29].

4.6.1 *Average Percentage of Fault Detection (APFD).*

APFD is a widely accepted metric for evaluating test prioritization effectiveness. A higher APFD value signifies greater effectiveness. APFD values are calculated using Formula 6.

$$APFD = 1 - \frac{\sum_{i=1}^{k} o_i}{kN} + \frac{1}{2N} \tag{6}$$

- N represents the total number of test inputs in the test set.
- k signifies the number of misclassified test inputs.
- o_i refers to the index of the i_{th} misclassified test within the prioritized test set.

Below, we explain from a mathematical perspective why a larger APFD indicates better effectiveness of a test prioritization method: Given that N is a constant, a higher APFD value corresponds to a smaller $\sum_{i=1}^{k} o_i$ (the index sum of misclassified tests in the prioritized list). A smaller $\sum_{i=1}^{k} o_i$ suggests that misclassified tests are positioned closer to the beginning of the prioritized test set, indicating that the test prioritization approach prioritized misclassified tests higher. Such an approach is considered to exhibit a high level of effectiveness.

Consistent with prior research [29], we normalize APFD values to the range [0, 1]. A prioritization approach is considered more effective if its APFD value approaches 1.

4.6.2 *Percentage of Fault Detected (PFD).*

PFD quantifies the ratio of correctly identified misclassified test inputs to the total count of misclassified tests. A higher PFD value indicates greater effectiveness of a test prioritization approach. PFD is calculated as shown in Formula 7.

$$PFD = \frac{F_c}{F_t} \tag{7}$$

- F_c represents the number of detected misclassified test inputs
- F_t is the total number of misclassified test inputs

  In our investigation, we evaluated the PFD values of PriCod across different ratios of prioritized tests. We use **PFD-n** to denote the first n% prioritized test inputs.

4.7 Im plementation and Configuration

We implemented PriCod using Python and the PyTorch 2.0.0 framework [78] and TensorFlow 2.3.1 framework. To facilitate comparisons with other approaches, we integrated existing implementations of the compared methods [29, 99] into our experimental pipeline. The compression of DNN models to the CoreML format was performed on macOS Ventura 13.4.1, as CoreML models can only be executed on iOS systems. For the classifier used in the ranking process, we employed LightGBM 3.3.5 with specific parameter settings: the learning rate of 0.1, *n_estimators* of 100, and *min_child_sample* of 20. Furthermore, we leveraged the packages SciPy 1.4.1 and scikit-learn 1.1.3 for data processing. Below, we present the test accuracy and training accuracy of the ranking model LightGBM on each dataset: **1) CIFAR10** Training Accuracy: 96.82%~97.84%; Test Accuracy: 77.42%~82.24% **2) CIDAR100** Training Accuracy: 96.28%~97.13%; Test Accuracy: 82.15%~85.58% **3) Fashion** Training Accuracy: 97.53%~97.96%; Test Accuracy: 86.31%~86.75% **4) Plant** Training Accuracy: 97.59%~98.12%; Test Accuracy: 85.51%~89.15% **5) News** Training Accuracy: 94.26%~96.13%; Test Accuracy: 77.79%~80.25%. In our experiments, the accuracy of the compressed DNN models used to evaluate PriCod ranged from 70.03% to 77.43%. The accuracy range of their original DNN model was between 70.23% and 78.74%. Other fundamental information about the models can be found in Table 2. Our experimental setup involved conducting experiments on NVIDIA Tesla V100 32GB GPUs. We used a MacBook Pro laptop running macOS Ventura 13.4.1 for data analysis, equipped with an Intel Core i9 CPU and 64 GB of RAM. In total, our study encompassed experiments involving 182 subjects, with 20 subjects based on natural inputs, 126 subjects based on noisy inputs, and 36 subjects based on adversarial inputs.

5 RESULTS AND ANALYSIS

5.1 RQ1: Performance of PriCod on Natural Test Inputs

Objectives: We evaluate the effectiveness of PriCod on natural test inputs with 20 compressed DNN models. We compare PriCod with a set of existing test prioritization approaches and a baseline method (i.e., random selection). Moreover, we evaluate PriCod on compressed DNN models with different accuracy levels, aiming to better assess the effectiveness of PriCod. Specifically, the experiments are conducted based on the following two sub-questions:

- **RQ-1.1** How does PriCod perform in terms of effectiveness and efficiency when applied to natural test inputs?
- **RQ-1.2** How does PriCod perform on compressed DNN models with different accuracy levels?

Experimental design: We conducted the following experiments to answer the aforementioned sub-questions, respectively.
[Experiment for RQ-1.1] We assessed the effectiveness and efficiency of PriCod on natural test inputs through the following experimental steps.

- **Subject Construction** We constructed 20 subjects consisting of compressed DNN models and their corresponding datasets. Specifically, we utilized 20 compressed DNN models along with three datasets. For specific details regarding the models and datasets, please refer to Section 4.2. The matching relationships are illustrated in Table 2.
- **Selection of Comparative Approaches** Subsequently, we meticulously selected five comparative approaches (i.e., DeepGini, Vanilla SM, PCS, entropy, and random selection) from the existing literature [29, 99]. These approaches can be adapted for prioritizing test inputs for compressed DNN models, with random selection as the baseline.
- **Evaluation of Effectiveness** Within our constructed 20 subjects, we evaluated the effectiveness of PriCod and all comparative methods using two widely adopted metrics: Average Percentage of Fault-Detection (APFD) and Percentage of Fault Detected (PFD) [29]. Detailed explanations of these metrics can be found in Section 4.6.
- **Comparison of Efficiency** Moreover, we compared the efficiency of PriCod and other test prioritization methods by analyzing their time costs.
- **Statistical Analysis** Recognizing the inherent variability within the model training process, we undertook a statistical analysis by executing each experiment ten times. We showcase the average outcomes of these trials.

More specifically, we employed the paired two-sample t-test [46], a widely utilized statistical method for comparing differences between two related datasets. The fundamental steps involve: 1) selecting two sets of related data, 2) calculating the difference for each corresponding pair of data points, and 3) analyzing these differences to assess whether there is a statistically significant disparity between the two datasets. In the context of the paired two-sample t-test, the significance of the results is determined by the p-value. Typically, a p-value less than 10^{-05} indicates a statistically significant difference between the two datasets [62].

[Experiment for RQ-1.2] We evaluated PriCod on compressed DNN models with varying accuracy levels using the following experimental steps. Initially, we trained a group of compressed DNN models at various accuracy levels, specifically 50%, 60%, 70%, 80%, and 90%. We evaluated the effectiveness of PriCod separately for each accuracy level using the APFD metric. Subsequently, we presented and compared PriCod's effectiveness across different accuracy levels through tabular representation.

Table 3. Effectiveness comparison among PriCod, DeepGini, VanillaSM, PCS, Entropy, Margin, LC, ATS and random selection in terms of the APFD values on natural test inputs

Data	Model	Approach								
		Random	DeepGini	VanillaSM	PCS	Entropy	Margin	LC	ATS	**PriCod**
CIFAR10	AlexNet-coreml	0.501	0.668	0.666	0.658	0.669	0.658	0.666	0.599	0.721
	AlexNet-tflite	0.502	0.670	0.668	0.660	0.671	0.660	0.668	0.601	0.720
	VGG16-coreml	0.497	0.748	0.747	0.744	0.747	0.744	0.747	0.708	0.781
	VGG16-tflite	0.501	0.750	0.749	0.747	0.748	0.747	0.749	0.707	0.781
CIFAR100	DenseNet201-coreml	0.502	0.737	0.737	0.734	0.735	0.734	0.737	0.712	0.788
	DenseNet201-tflite	0.501	0.753	0.755	0.753	0.747	0.753	0.755	0.703	0.796
	ResNet152-coreml	0.498	0.710	0.711	0.707	0.703	0.707	0.711	0.688	0.765
	ResNet152-tflite	0.496	0.749	0.751	0.746	0.741	0.746	0.751	0.691	0.786
Fashion	LeNet1-coreml	0.502	0.743	0.744	0.742	0.737	0.742	0.744	0.616	0.815
	LeNet1-tflite	0.504	0.743	0.744	0.741	0.737	0.741	0.744	0.615	0.815
	LeNet5-coreml	0.508	0.763	0.763	0.757	0.760	0.757	0.763	0.623	0.826
	LeNet5-tflite	0.496	0.763	0.763	0.757	0.760	0.757	0.763	0.626	0.824
Plant	NIN-coreml	0.501	0.740	0.743	0.743	0.736	0.743	0.743	0.711	0.795
	NIN-tflite	0.501	0.742	0.744	0.744	0.737	0.744	0.744	0.714	0.794
	VGG19-coreml	0.497	0.687	0.685	0.683	0.687	0.683	0.685	0.643	0.779
	VGG19-tflite	0.502	0.688	0.687	0.684	0.689	0.684	0.687	0.645	0.781
News	GRU-coreml	0.491	0.713	0.715	0.713	0.707	0.713	0.715	0.684	0.756
	GRU-tflite	0.505	0.713	0.715	0.712	0.707	0.712	0.715	0.685	0.757
	LSTM-coreml	0.503	0.729	0.731	0.732	0.723	0.732	0.731	0.691	0.771
	LSTM-tflite	0.511	0.729	0.732	0.733	0.723	0.733	0.732	0.692	0.770

Table 4. Average improvement of PriCod over the compared approaches in terms of the APFD values on natural test inputs

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.501	55.89
DeepGini	0	0.726	7.58
VanillaSM	0	0.727	7.43
PCS	0	0.724	7.87
Entropy	0	0.723	8.02
Margin	0	0.724	7.87
LC	0	0.727	7.43
ATS	0	0.668	16.92
PriCod	20	0.781	-

Results: The experimental results for RQ-1.1 are depicted in Table 3, Table 4, Table 5, Table 6, and Table 7. Among these, the first two tables compare PriCod and other test prioritization methods based on the APFD metric using

Table 5. Average comparison results among PriCod and the compared approaches in terms of PFD on natural test inputs

Data	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
	Random	0.105	0.201	0.302	0.402	0.498	0.599	0.699
	DeepGini	0.230	0.418	0.577	0.713	0.819	0.896	0.945
	VanillaSM	0.230	0.414	0.575	0.710	0.818	0.895	0.945
	PCS	0.213	0.403	0.568	0.705	0.814	0.893	0.943
CIFAR10	Entropy	0.226	0.415	0.575	0.714	0.821	0.898	0.947
	Margin	0.213	0.403	0.568	0.705	0.814	0.893	0.943
	LC	0.230	0.414	0.575	0.710	0.818	0.895	0.945
	ATS	0.206	0.367	0.503	0.612	0.711	0.807	0.901
	PriCod	0.273	0.489	0.661	0.786	0.875	0.935	0.971
	Random	0.102	0.199	0.299	0.402	0.501	0.601	0.699
	DeepGini	0.249	0.459	0.632	0.769	0.869	0.935	0.972
	VanillaSM	0.253	0.462	0.635	0.771	0.870	0.935	0.972
	PCS	0.241	0.452	0.630	0.769	0.869	0.934	0.972
CIFAR100	Entropy	0.245	0.449	0.619	0.756	0.859	0.929	0.971
	Margin	0.241	0.452	0.630	0.769	0.869	0.934	0.972
	LC	0.253	0.462	0.635	0.771	0.870	0.935	0.972
	ATS	0.221	0.416	0.581	0.702	0.784	0.854	0.913
	PriCod	0.283	0.521	0.711	0.847	0.931	0.973	0.991
	Random	0.102	0.197	0.299	0.397	0.501	0.599	0.699
	DeepGini	0.258	0.479	0.665	0.805	0.893	0.944	0.974
	VanillaSM	0.259	0.483	0.666	0.804	0.893	0.944	0.973
	PCS	0.248	0.468	0.659	0.801	0.891	0.942	0.974
Fashion	Entropy	0.253	0.472	0.654	0.791	0.890	0.942	0.974
	Margin	0.248	0.468	0.659	0.801	0.891	0.942	0.974
	LC	0.259	0.483	0.666	0.804	0.893	0.944	0.973
	ATS	0.219	0.344	0.425	0.521	0.612	0.713	0.864
	PriCod	0.356	0.626	0.812	0.919	0.968	0.988	0.996
	Random	0.101	0.199	0.298	0.399	0.502	0.603	0.703
	DeepGini	0.219	0.412	0.575	0.718	0.834	0.911	0.963
	VanillaSM	0.221	0.409	0.578	0.721	0.834	0.912	0.963
	PCS	0.214	0.406	0.577	0.722	0.834	0.913	0.964
Plant	Entropy	0.217	0.410	0.576	0.713	0.828	0.909	0.961
	Margin	0.214	0.406	0.577	0.722	0.834	0.913	0.964
	LC	0.221	0.409	0.578	0.721	0.834	0.912	0.963
	ATS	0.201	0.362	0.523	0.657	0.763	0.851	0.912
	PriCod	0.281	0.531	0.735	0.876	0.949	0.983	0.994
	Random	0.095	0.194	0.291	0.393	0.493	0.595	0.695
	DeepGini	0.239	0.446	0.618	0.740	0.833	0.898	0.946
	VanillaSM	0.249	0.451	0.623	0.739	0.833	0.897	0.946
	PCS	0.243	0.453	0.623	0.738	0.834	0.897	0.945
News	Entropy	0.233	0.429	0.598	0.732	0.830	0.895	0.947
	Margin	0.243	0.453	0.623	0.738	0.834	0.897	0.945
	LC	0.249	0.451	0.623	0.739	0.833	0.897	0.946
	ATS	0.212	0.415	0.531	0.672	0.725	0.801	0.887
	PriCod	0.281	0.501	0.666	0.791	0.864	0.926	0.965

natural test inputs. Table 5 presents a comparison using the PFD metric. Table 6 presents detailed results from the statistical analysis in terms of PFD. Table 7 illustrates the comparison in terms of efficiency.

Table 6. Statistical analysis on natural test inputs (in terms of p-value on PFD)

	Approach							
	Random	DeepGini	VanillaSM	PCS	Entropy	Margin	LC	ATS
Pricod	3.88×10^{-14}	3.22×10^{-6}	1.21×10^{-7}	6.69×10^{-6}	5.12×10^{-6}	6.69×10^{-6}	1.21×10^{-7}	2.53×10^{-9}

Table 7. Time cost of PriCod and the compared test prioritization approaches

Time cost	Approach								
	PriCod	Random	DeepGini	VanillaSM	PCS	Entropy	Margin	LC	ATS
Feature generation	9 min	-	-	-	-	-	-	-	-
Ranking model training	18 s	-	-	-	-	-	-	-	-
Prediction	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	>10 min

PriCod consistently outperforms all the compared approaches (i.e., DeepGini, Vanilla SM, PCS, Entropy, and Random) in terms of effectiveness. Table 3 showcases the effectiveness comparison of PriCod and all comparative methods across different subjects, measured using the APFD metric. Notably, we highlighted the approach with the highest effectiveness in gray for each case. From Table 3, we see that PriCod performs better than all the comparative methods across all subjects. The range of PriCod's APFD values spans from 0.720 to 0.826, while the APFD values for the comparative methods fall within the range of 0.491 to 0.763. Table 4 delves deeper into the effectiveness comparison between PriCod and other test prioritization methods. This comparison is approached from three perspectives: the number of cases in which each approach performs the best, the average APFD of each method, and the average improvement in APFD of PriCod over the compared test prioritization methods. Notably, the average APFD value achieved by PriCod is 0.781, with an average improvement ranging from 7.43% to 55.89% over the compared test prioritization approaches. Table 5 displays the comparative results between PriCod and existing test prioritization methods utilizing the metric PFD. We see that PriCod showcases a higher level of effectiveness across different prioritized test ratios, surpassing all the compared techniques. The aforementioned observations strongly demonstrate that PriCod outperforms all the compared approaches in terms of both APFD and PFD.

As mentioned in the experimental design, we conducted a statistical analysis to evaluate the stability of our findings. This involved repeating all experiments ten times. All results presented are the average values obtained from these ten repetitions. Furthermore, we identified that the p-value is less than 10^{-05}, underscoring the consistent superiority of PriCod over the compared methods in test prioritization.

Moreover, Table 6 presents detailed results from the statistical analysis in terms of PFD. We see that all the p-values between PriCod and the compared approaches consistently fall below 10^{-05}, indicating that PriCod statistically outperforms all the compared methods in terms of PFD. For example, the p-value for the difference in experimental results between PriCod and DeepGini is 3.22×10^{-06}. The p-value between PriCod and PCS is 6.69×10^{-06}.

The efficiency of PriCod falls within an acceptable range. Table 7 presents a comparison of the efficiency between PriCod and the compared methods. We observe that PriCod's total execution time is less than 9 min 20s, which can be divided into three main components: Feature generation, training, and prediction. Among these, feature generation takes 9 minutes, training requires 18 seconds, and prediction is completed in less than 1 second. The final prediction time for the compared methods is less than 1 second. While PriCod is not as efficient as the confidence-based test prioritization approaches, its efficiency falls within an acceptable range.

Answer to RQ1.1: *PriCod consistently outperforms all the compared approaches (i.e., DeepGini, Vanilla SM, PCS, Entropy, and Random), with an average improvement of 8.28% to 56.69% in terms of APFD. Moreover, the efficiency of PriCod falls within an acceptable range.*

The experimental results for RQ-1.2 are displayed in Table 8. In each case, we highlighted the approach with the highest effectiveness in gray. In Table 8, we see that, across compressed DNN models at different accuracy levels, PriCod consistently exhibits the highest effectiveness, as measured by APFD. Specifically, PriCod achieves an average APFD of 0.793 across all accuracy levels, while the average APFD for the comparison methods ranges from 0.498 to 0.732. These experimental findings indicate that, across models with different accuracy levels, PriCod's effectiveness surpasses all compared testing prioritization methods.

The methods used for comparison, including DeepGini, VanillaSM, PCS, Entropy, Margin, LC, and ATS, are influenced by the accuracy of the compressed models. For instance, DeepGini exhibits an APFD of 0.862 in compressed models with 90% accuracy, while it decreases to 0.604 in compressed models with 50% accuracy. Similarly, PCS has an APFD of 0.861 in compressed models with 90% accuracy, and it decreases to 0.595 in compressed models with 50% accuracy. In contrast, PriCod's performance is relatively less affected by the accuracy of the compressed models compared to these methods. For instance, in compressed models with 90% accuracy, PriCod has an APFD of 0.864. In compressed models with 50% accuracy, its APFD is 0.713.

Answer to RQ1.2: *Across compressed DNN models with varying accuracy levels, PriCod consistently performs better than all the compared test prioritization methods.*

Table 8. Average Effectiveness comparison among PriCod, DeepGini, VanillaSM, PCS, Entropy, Margin, LC, ATS, and random selection in terms of the APFD values on different accuracy compressed models

Approach	**Accuracy**					**Average APFD**
	50%	60%	70%	80%	90%	
Random	0.501	0.499	0.497	0.501	0.493	0.498
DeepGini	0.604	0.658	0.739	0.784	0.862	0.729
VanillaSM	0.604	0.664	0.742	0.786	0.863	0.732
PCS	0.595	0.656	0.736	0.782	0.861	0.726
Entropy	0.603	0.648	0.732	0.780	0.859	0.724
Margin	0.595	0.656	0.736	0.782	0.861	0.726
LC	0.604	0.664	0.742	0.786	0.863	0.732
ATS	0.586	0.623	0.661	0.735	0.806	0.682
PriCod	0.713	0.751	0.801	0.839	0.864	0.793

5.2 RQ2: Effectiveness on Noisy Test Inputs

Objectives: When deployed on mobile devices, compressed DNN models can encounter noisy data due to various factors. These factors encompass user behaviors, such as capturing photos from various angles. All these elements have the potential to introduce noise into the images. As a result, it becomes crucial to evaluate the effectiveness of PriCod using noisy datasets. To accomplish this, we utilize 13 noise generation techniques [69, 80, 85, 89] to construct datasets with inherent noise for the purpose of assessment.
Experimental design: We introduce noise to the original datasets using 13 noise generation techniques collected from existing literature. Specifically, given an original test set, we extract 30% of the tests and transform them into noisy data using a noise technique, while the remaining 70% are left unchanged. The comparative methods we employed in this research question, along with the evaluation metrics, remained consistent with RQ1. The comparative methods were DeepGini, Vanilla SM, PCS, entropy, and random selection. Moreover, we utilized the APFD and PFD metrics (cf. Section 4.6) to evaluate the effectiveness of PriCod.

Table 9. Average Effectiveness comparison among PriCod, DeepGini, VanillaSM, PCS, Entropy, Margin, LC, ATS, and random selection in terms of the APFD values on noisy test inputs

Noisy Techniques	**Approach**								
	Random	DeepGini	VanillaSM	PCS	Entropy	Margin	LC	ATS	**PriCod**
CSR	0.501	0.724	0.724	0.720	0.721	0.720	0.724	0.645	0.775
FSN	0.499	0.729	0.729	0.724	0.726	0.724	0.729	0.650	0.779
HS	0.499	0.708	0.708	0.703	0.706	0.703	0.708	0.641	0.767
HF	0.499	0.693	0.692	0.687	0.692	0.687	0.692	0.637	0.762
VF	0.500	0.675	0.673	0.667	0.675	0.667	0.673	0.609	0.737
RO	0.499	0.689	0.688	0.681	0.690	0.681	0.688	0.626	0.753
SR	0.499	0.729	0.729	0.724	0.726	0.724	0.729	0.649	0.779
WS	0.497	0.711	0.710	0.704	0.710	0.704	0.710	0.634	0.771
ZCA	0.500	0.729	0.729	0.724	0.726	0.724	0.729	0.651	0.778
ZOO	0.501	0.684	0.681	0.674	0.686	0.674	0.681	0.608	0.748
CON	0.500	0.704	0.703	0.698	0.703	0.698	0.703	0.672	0.746
GAS	0.499	0.699	0.696	0.689	0.700	0.689	0.696	0.614	0.759
SP	0.500	0.711	0.711	0.706	0.709	0.706	0.711	0.638	0.771

Table 10. Average improvement of PriCod over the compared approaches in terms of the APFD values on noisy test inputs

Approach	**# Best cases**	**Average APFD**	**Improvement(%)**
Random	0	0.499	52.91
DeepGini	0	0.707	7.92
VanillaSM	0	0.706	8.07
PCS	0	0.701	8.84
Entropy	0	0.705	8.23
Margin	0	0.701	8.84
LC	0	0.706	8.07
ATS	0	0.636	19.97
PriCod	126	0.763	-

Results: The experimental results for RQ2 are presented in Table 9, Table 10, Table 11, and Table 12. When applied to noisy test inputs, PriCod consistently exhibits superior performance compared to all the test prioritization approaches under different noise generation techniques. Specifically, Table 9 and Table 10 illustrate the effectiveness of PriCod and the compared test prioritization methods based on the APFD metric. We see that PriCod's APFD values range between 0.737 and 0.779, while the compared test prioritization methods range from 0.497 and 0.729. Furthermore, Table 10 offers a more comprehensive analysis, showcasing the best cases achieved by each approach, the average APFD value of each method, and PriCod's effectiveness improvement relative to each comparative method. Notably, PriCod demonstrates the highest effectiveness across all cases. The average APFD value achieved by PriCod is 0.763, while that of the compared approaches falls between 0.499 and 0.707. The improvement achieved by PriCod over the comparative methods varies from 7.92% to 52.91%.

Table 11 and Table 12 present a comprehensive comparative analysis of PriCod and the compared approaches regarding the PFD metric. In Table 11, we exhibit experimental results under seven noises, while the results for other noise scenarios can be found on our GitHub[2]. In Table 11, we see that, across different noisy techniques, PriCod consistently outperforms the compared approaches in terms of PFD. Moreover, Table 12 exhibits that PriCod performs the best across varying proportions of prioritized tests. Notably, when prioritizing 50% of the tests, PriCod can identify 90.3% of misclassified tests, while the competing methods can only identify 50.1% to 81.9% misclassified tests. These experimental results demonstrate that PriCod performs better than all the compared test prioritization methods when applied to noisy test inputs.

[2]https://github.com/yinghuali/PriCod/tree/main/tables

Table 11. Effectiveness comparison results among PriCod and the compared approaches in terms of PFD on noisy test inputs

Noisy Techniques	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
CSR	Random	0.100	0.199	0.301	0.403	0.502	0.601	0.702
	DeepGini	0.236	0.436	0.604	0.743	0.846	0.915	0.958
	Entropy	0.232	0.428	0.599	0.736	0.843	0.914	0.959
	PCS	0.226	0.426	0.597	0.737	0.843	0.913	0.957
	VanillaSM	0.238	0.437	0.604	0.742	0.845	0.915	0.958
	Margin	0.226	0.426	0.597	0.737	0.843	0.913	0.957
	LC	0.238	0.437	0.604	0.742	0.845	0.915	0.958
	ATS	0.207	0.352	0.458	0.559	0.654	0.755	0.845
	PriCod	0.299	0.539	0.721	0.845	0.918	0.961	0.983
FSN	Random	0.101	0.200	0.299	0.400	0.501	0.603	0.702
	DeepGini	0.242	0.445	0.616	0.754	0.853	0.918	0.960
	Entropy	0.238	0.439	0.609	0.748	0.851	0.917	0.960
	PCS	0.229	0.433	0.609	0.748	0.850	0.917	0.959
	VanillaSM	0.243	0.445	0.616	0.753	0.852	0.918	0.960
	Margin	0.229	0.433	0.609	0.748	0.850	0.917	0.959
	LC	0.243	0.445	0.616	0.753	0.852	0.918	0.960
	ATS	0.212	0.355	0.464	0.566	0.661	0.759	0.850
	PriCod	0.306	0.547	0.730	0.850	0.923	0.963	0.985
HS	Random	0.099	0.199	0.300	0.399	0.498	0.601	0.701
	DeepGini	0.220	0.407	0.571	0.711	0.823	0.902	0.953
	Entropy	0.217	0.401	0.564	0.706	0.818	0.901	0.952
	PCS	0.205	0.394	0.562	0.704	0.817	0.899	0.951
	VanillaSM	0.218	0.407	0.571	0.709	0.822	0.901	0.953
	Margin	0.205	0.394	0.562	0.704	0.817	0.899	0.951
	LC	0.218	0.407	0.571	0.709	0.822	0.901	0.953
	ATS	0.196	0.347	0.458	0.556	0.647	0.744	0.837
	PriCod	0.278	0.503	0.687	0.818	0.904	0.953	0.980
HF	Random	0.099	0.200	0.300	0.401	0.499	0.599	0.699
	DeepGini	0.218	0.403	0.562	0.697	0.799	0.870	0.921
	Entropy	0.216	0.399	0.559	0.694	0.799	0.871	0.923
	PCS	0.204	0.391	0.555	0.689	0.793	0.867	0.918
	VanillaSM	0.217	0.402	0.562	0.695	0.798	0.869	0.921
	Margin	0.204	0.391	0.555	0.689	0.793	0.867	0.918
	LC	0.217	0.402	0.562	0.695	0.798	0.869	0.921
	ATS	0.196	0.344	0.459	0.560	0.652	0.745	0.830
	PriCod	0.287	0.517	0.698	0.829	0.910	0.957	0.982
RO	Random	0.100	0.200	0.299	0.399	0.498	0.597	0.697
	DeepGini	0.204	0.382	0.541	0.679	0.792	0.876	0.934
	Entropy	0.202	0.382	0.543	0.681	0.793	0.877	0.936
	PCS	0.189	0.365	0.525	0.666	0.784	0.872	0.932
	VanillaSM	0.202	0.378	0.536	0.675	0.790	0.875	0.934
	Margin	0.189	0.365	0.525	0.666	0.784	0.872	0.932
	LC	0.202	0.378	0.536	0.675	0.790	0.875	0.934
	ATS	0.180	0.326	0.440	0.534	0.626	0.726	0.826
	PriCod	0.258	0.474	0.655	0.794	0.888	0.946	0.977
SR	Random	0.099	0.201	0.301	0.400	0.501	0.600	0.700
	DeepGini	0.241	0.445	0.615	0.755	0.853	0.918	0.960
	Entropy	0.237	0.439	0.609	0.747	0.851	0.917	0.960
	PCS	0.228	0.433	0.608	0.748	0.849	0.917	0.958
	VanillaSM	0.242	0.445	0.615	0.753	0.852	0.918	0.960
	Margin	0.228	0.433	0.608	0.748	0.849	0.917	0.958
	LC	0.242	0.445	0.615	0.753	0.852	0.918	0.960
	ATS	0.211	0.355	0.464	0.565	0.660	0.758	0.849
	PriCod	0.305	0.547	0.730	0.850	0.922	0.963	0.985
VF	Random	0.099	0.200	0.300	0.401	0.500	0.601	0.701
	DeepGini	0.188	0.358	0.512	0.652	0.771	0.863	0.927
	Entropy	0.186	0.358	0.513	0.654	0.772	0.865	0.929
	PCS	0.175	0.339	0.496	0.639	0.761	0.857	0.924
	VanillaSM	0.187	0.354	0.508	0.648	0.769	0.862	0.927
	Margin	0.175	0.339	0.496	0.639	0.761	0.857	0.924
	LC	0.187	0.354	0.508	0.648	0.769	0.862	0.927
	ATS	0.167	0.306	0.416	0.510	0.604	0.705	0.807
	PriCod	0.233	0.442	0.619	0.764	0.868	0.935	0.972

Table 12. Average effectiveness comparison results among PriCod and the compared approaches in terms of PFD on noisy data

Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
Random	0.101	0.199	0.301	0.398	0.501	0.602	0.701
DeepGini	0.220	0.410	0.573	0.711	0.819	0.895	0.946
VanillaSM	0.220	0.408	0.571	0.709	0.818	0.895	0.946
Entropy	0.217	0.406	0.570	0.709	0.818	0.896	0.947
PCS	0.207	0.395	0.562	0.702	0.812	0.892	0.944
Margin	0.207	0.395	0.562	0.702	0.812	0.892	0.944
LC	0.220	0.408	0.571	0.709	0.818	0.895	0.946
ATS	0.193	0.338	0.449	0.550	0.644	0.743	0.837
PriCod	0.277	0.506	0.687	0.818	0.903	0.953	0.981

Answer to RQ2: *When applied to noisy test inputs, PriCod continues to outperform all the compared approaches in terms of both APFD (Average Percentage of Fault Detection) and PFD (Percentage of Fault Detection). The improvement achieved by PriCod over the comparative methods varies from 8.46% to 53.80% in terms of APFD.*

5.3 RQ3: Effectiveness on Adversarial Test Inputs

Objectives: Besides evaluating the effectiveness of PriCod on natural and noisy test inputs, following the evaluation methodology of previous test prioritization research [97], we also assess its effectiveness on adversarial test inputs.

Experimental design: To generate adversarial test samples, we utilized four distinct adversarial attack techniques: the Fast Gradient Method (FGM) [32], Adversarial Patch (Patch) [5], Basic Iterative Method (BIM) [50], and Projected Gradient Descent (PGD) [63]. This yielded a set of 32 subjects for evaluation. Consistent with our prior research questions, we conducted a comparative analysis between PriCod and four alternative test prioritization approaches, along with a baseline method (random selection). The effectiveness of these methods was measured using the metrics APFD and PFD.

Table 13. Average Effectiveness comparison among PriCod, DeepGini, VanillaSM, PCS, Entropy, Margin, LC, ATS and random selection in terms of the APFD values on adversarial test inputs

Adversarial Attack Techniques	Approach								
	Random	DeepGini	VanillaSM	PCS	Entropy	Margin	LC	ATS	**PriCod**
BIM	0.499	0.727	0.727	0.722	0.725	0.722	0.727	0.654	0.773
FGM	0.501	0.729	0.731	0.724	0.727	0.724	0.731	0.658	0.776
Patch	0.499	0.661	0.660	0.656	0.659	0.656	0.660	0.617	0.721
PGD	0.501	0.725	0.725	0.721	0.722	0.721	0.725	0.652	0.771

Table 14. Average improvement of PriCod over the compared approaches in terms of the APFD values on adversarial test inputs

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.501	51.59
DeepGini	0	0.710	7.18
VanillaSM	0	0.711	7.03
PCS	0	0.705	7.94
Entropy	0	0.708	7.49
Margin	0	0.705	7.94
LC	0	0.711	7.03
ATS	0	0.645	17.98
PriCod	36	0.761	-

Table 15. Average effectiveness comparison results among PriCod and the compared approaches on adversarial test inputs in terms of PFD

Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
Random	0.099	0.199	0.298	0.399	0.501	0.599	0.701
DeepGini	0.224	0.417	0.582	0.719	0.822	0.895	0.944
Entropy	0.220	0.412	0.576	0.713	0.819	0.894	0.944
PCS	0.213	0.406	0.575	0.713	0.818	0.893	0.943
VanillaSM	0.225	0.416	0.582	0.718	0.821	0.894	0.944
Margin	0.213	0.406	0.575	0.713	0.818	0.893	0.943
LC	0.225	0.416	0.582	0.718	0.821	0.894	0.944
ATS	0.201	0.346	0.459	0.561	0.655	0.754	0.846
PriCod	0.282	0.511	0.690	0.816	0.896	0.946	0.975

Results: The experimental results for RQ3 are presented in Table 13, Table 14, and Table 15. Among these, Table 13 and Table 14 illustrate the comparative results between PriCod and the compared test prioritization approaches based on the APFD metric on the adversarial test inputs. Table 15 showcases the comparative results based on the PFD metric. From Table 13, we see that across different attack techniques, PriCod consistently exhibits the highest effectiveness, with the range of its average APFD scores lying between 0.721 and 0.776. In contrast, the average scores of the compared methods range from 0.499 and 0.731. Within Table 14, we see that PriCod outperforms all other test prioritization approaches across all 32 cases. PriCod's average APFD score across all subjects is 0.761, while that of the compared methods ranges from 0.501 and 0.711. Furthermore, PriCod achieves an improvement of 7.03% to 51.59% over all the compared methods.

Table 15 illustrates the comparison of effectiveness between PriCod and other test prioritization methods based on the PFD metric. Notably, PriCod consistently demonstrates superior effectiveness across varying test prioritization ratios. Notably, when 50% of the tests are prioritized, PriCod can identify 89.6% of misclassified tests. In contrast, the compared methods only managed to identify 50.1% to 82.2% of misclassified tests under the same conditions. Additionally, with a prioritization of 40% of the tests, PriCod can identify 81.6% of misclassified tests, whereas the compared methods only achieve a range of 39.9% to 71.9% for the same metric. These results collectively demonstrate that in terms of the APFD and PFD metrics, PriCod's effectiveness surpasses that of all compared test prioritization methods.

Answer to RQ3: *When applied to adversarial test inputs, PriCod continues to outperform all the compared test prioritization approaches in terms of both APFD and PFD. PriCod achieves an improvement of 8.24%~55.31% over all the compared approaches.*

5.4 RQ4: Impact of fusion strategies

Objectives: We conducted an in-depth study of the impact of different feature fusion methods on the effectiveness of PriCod.

Experimental design: To investigate the impact of different feature fusion methods on the effectiveness of PriCod, we designed three variants of PriCod. Each variant employs a distinct feature fusion approach to combine the embedding features and deviation features. These three variants are denoted as PriCod^a, PriCod^m, and PriCod^c, which respectively utilize addition, multiplication, and cross-multiplication techniques for feature fusion. Apart from the method of feature fusion, the rest of these variants remain consistent with the original PriCod. Subsequently, we evaluated their effectiveness along with the original PriCod on natural datasets. This comparison enabled us to assess the influence of various fusion methods on the effectiveness of PriCod.

Below, we explain why the studied feature fusion strategies are meaningful.

- **Addition-based feature fusion strategy** The addition-based fusion strategy is meaningful due to several reasons: 1) The addition-based fusion strategy preserves the original information of each feature; 2) the addition-based fusion strategy is simple and efficient to implement; 3) the addition-based fusion strategy is suitable for cases where two sets of features are relatively independent, with each contributing independently to the prediction results.
- **Multiplication-based feature fusion strategy** The multiplication-based fusion strategy is meaningful for various reasons: 1) The utilization of the multiplication operation introduces non-linear transformations, enhancing the capability of the ranking model to grasp more feature relationships. 2) Incorporating the multiplication operation aids in mitigating the influence of irrelevant features. When the value of a feature is small, the multiplication diminishes its overall contribution, thereby reducing its impact.
- **Cross-multiplication-based feature fusion strategy** The cross-multiplication-based fusion strategy is meaningful for various reasons: 1) Cross-multiplication-based operations can introduce stronger information interaction between features. This can help the model better understand the relationships between features, thereby enhancing its representational capacity. 2) By considering all possible pairwise interactions, the model operates in a higher-dimensional feature space, capable of revealing more hidden patterns and relationships.

Table 16. Effectiveness comparison among PriCod and PriCod Variants in terms of the APFD values on natural test inputs

Data	Model	Approach			
		PriCod^a	PriCod^m	PriCod^c	**PriCod**
CIFAR10	AlexNet-coreml	0.572	0.689	0.717	0.721
	AlexNet-tflite	0.573	0.686	0.717	0.720
	VGG16-coreml	0.611	0.756	0.780	0.781
	VGG16-tflite	0.610	0.756	0.778	0.781
CIFAR100	DenseNet201-coreml	0.654	0.762	0.785	0.788
	DenseNet201-tflite	0.651	0.752	0.782	0.796
	ResNet152-coreml	0.658	0.749	0.764	0.765
	ResNet152-tflite	0.662	0.751	0.776	0.786
Fashion	LeNet1-coreml	0.755	0.783	0.808	0.815
	LeNet1-tflite	0.754	0.771	0.807	0.815
	LeNet5-coreml	0.760	0.794	0.820	0.826
	LeNet5-tflite	0.759	0.783	0.817	0.824
Plant	NIN-coreml	0.671	0.760	0.793	0.795
	NIN-tflite	0.670	0.757	0.792	0.794
	VGG19-coreml	0.652	0.752	0.776	0.779
	VGG19-tflite	0.652	0.753	0.780	0.781
News	GRU-coreml	0.730	0.719	0.736	0.756
	GRU-tflite	0.726	0.717	0.737	0.757
	LSTM-coreml	0.744	0.734	0.754	0.771
	LSTM-tflite	0.744	0.731	0.754	0.770
	Average	0.681	0.748	0.774	0.781

Results: The experimental results for RQ4 are presented in Table 16. We have shaded the approach with the highest effectiveness in gray for each case. From the table, we see that the effectiveness of PriCod remains consistently highest across different subjects. Its APFD values range from 0.720 to 0.826. The variant PriCod^a exhibits an APFD range of 0.572 to 0.760. The variant PriCod^m demonstrates an APFD range of 0.686 to 0.794.

The variant PriCod^c shows an APFD range of 0.717 to 0.820. We see that the effectiveness of the PriCod^c variant is second only to the original PriCod. In our experiments, PriCod employs a concatenation approach for feature fusion. This indicates that, compared to addition, multiplication, and cross-multiplication, the concatenation method is more suitable for fusing the deviation features and embedding features of compressed DNN models for test prioritization.

Answer to RQ4: *The effectiveness of PriCod surpasses all variants, indicating that among all feature fusion methods, concatenation is more effective in combining the deviation features and embedding features of compressed DNN models for test prioritization.*

5.5 RQ5: Feature contribution analysis

Objectives: We delve into understanding the impact of different feature types on the effectiveness of PriCod in test prioritization. Our exploration centers around two key sub-questions presented below:

- **RQ-5.1** To what extent does each type of features contribute to the effectiveness of PriCod?
- **RQ-5.2** How are the feature types distributed among the top-N most influential features towards the effectiveness of PriCod?

Experimental design: We conducted two experiments to address the aforementioned sub-questions.
[Experiments for RQ-5.1] Within the initial PriCod framework, we generated two distinct categories of features: deviation features and embedding features. In order to assess the individual impact of each feature type on the effectiveness of PriCod, we conducted a carefully designed ablation study, following established methodologies as outlined in previous work [24]. Specifically, we excluded one feature type at a time while retaining the other. To elaborate, for the assessment of the contribution made by deviation features, we executed PriCod without incorporating deviation features while keeping the embedding features present. Conversely, to gauge the contribution of embedding features, we ran PriCod without embedding features but retained the deviation features. This meticulous ablation study facilitated a quantitative analysis of the influence exerted by each feature type on the overall effectiveness of PriCod.
[Experiments for RQ-5.2] To investigate the distribution of different feature types within the top N contributing features, we leveraged the cover metric of the XGBoost algorithm [12]. The specific experimental procedures are detailed below:

❶ **Feature Importance Calculation** Initially, we employed the cover metric to calculate the importance scores of each feature utilized by PriCod in the context of test prioritization.
❷ **Top-N Feature Selection** Subsequently, we identified the N most important features based on the computed scores.
❸ **Categorization Analysis** Through an analysis of the categorization of these selected features, we delved into the extent to which different feature types contribute to the effectiveness of PriCod.

We provide an outline of how XGBoost measures the importance of features as follows. Within the XGBoost algorithm, the cover metric serves as a fundamental tool for quantifying feature importance. This metric functions by assessing the average coverage of individual instances across the leaf nodes in decision trees. Essentially, the cover metric evaluates how frequently a specific feature is employed to partition data across all trees within the ensemble. The coverage values assigned to each feature across the entirety of trees are then combined, resulting in a cumulative coverage value. To determine the average coverage of each instance by the leaf nodes, the cumulative coverage value is normalized relative to the total number of instances. Consequently, the coverage value attributed to a particular feature plays a decisive role in establishing its significance within the context of the XGBoost model. Notably, features that exhibit higher coverage values are granted increased importance, shaping the XGBoost algorithm's decision-making process. This systematic approach empowers XGBoost to effectively assess the impact of individual features, contributing to accurate predictions and well-informed decisions.

Furthermore, we conducted a more detailed comparison to assess the contributions of different deviation features on the effectiveness of PriCod. Specifically, we performed comparative experiments through the following process.

❶ Under each subject, we calculate the importance value of each type of deviation feature, which reflects its impact on the effectiveness of PriCod within that specific subject.
❷ For each type of deviation feature, we calculated the sum of its importance values across all subjects. For example, in the case of the CLA feature, we summed up all its importance values across all subjects to obtain the final value.
❸ We normalized the final importance values of all deviation features to compare their contributions. We represented the results in the form of a pie chart. In the pie chart, if the proportion of a deviation feature is higher compared to others, it implies that this feature contributes more to the effectiveness of PriCod.

Table 17. Ablation study on different features of PriCod: Embedding Features(EB), Deviation Features (DF). 'w/o' means 'without'

Approach	Datasets					Average
	CIFAR10	CIFAR100	Fashion	Plant	News	
PriCod w/o EB	0.745	0.766	0.791	0.762	0.743	0.761
PriCod w/o DF	0.598	0.615	0.778	0.648	0.618	0.652
PriCod	0.751	0.784	0.820	0.787	0.763	0.781

Results: The experimental results for RQ5.1 are presented in Table 17. In this table, 'w/o' stands for 'without.' For instance, 'PriCod w/o EB' indicates the execution of PriCod without generating the embedding features. We highlighted the approach with the highest effectiveness in gray for each case.

Each type of features (i.e., deviation features and embedding features) contribute to the effectiveness of PriCod. As indicated by the results in Table 17, the unaltered PriCod model achieves the highest average effectiveness. Notably, the removal of any feature type leads to a reduction in PriCod's effectiveness, highlighting that each type of features plays a role in PriCod's effectiveness. For example, on the CIFAR10 dataset, the original PriCod attains an average APFD value of 0.751. Removing embedding features leads to a decrease in PriCod's average APFD to 0.745, while the absence of deviation features results in a more significant decline to 0.598.

Deviation features make the highest average contributions. Moreover, as indicated in Table 17, deviation features demonstrate the most substantial average contributions to the effectiveness of PriCod. Across all datasets, the impact of removing deviation features on PriCod's effectiveness is the most pronounced. On average, across all cases, the exclusion of deviation features leads to a decrease in APFD of 0.129, while the removal of embedding features only results in a decrease of 0.020. To provide specific examples, consider the CIFAR10 dataset: removing deviation features causes a noteworthy reduction in APFD by 0.153, compared to a minor decrease of 0.006 from removing embedding features. On the CIFAR100 dataset, the removal of deviation features results in a decrease in APFD by 0.169, in contrast to a marginal decrease of 0.018 observed when removing embedding features. Likewise, on the Fashion dataset, the absence of deviation features leads to a drop in APFD by 0.042, whereas removing embedding features leads to a decrease of 0.029. Similarly, on the Plant dataset, removing deviation features causes an APFD decrease of 0.139, whereas removing embedding features results in a decrease of 0.025.

Answer to RQ5.1: *Each type of features (i.e., deviation features and embedding features) contribute to the effectiveness of PriCod. Notably, deviation features make the highest average contributions.*

Table 18. Top-10 most contributing features on the effectiveness of PriCod: Embedding Features(EB), Deviation Features (DF)

Data	**Rank**	**AlexNet-coreml**		**AlexNet-tflite**		**VGG16-coreml**		**VGG16-tflite**	
		Feature	Value	Feature	Value	Feature	Value	Feature	Value
CIFAR10	1	DF-0	3022	DF-0	4486	DF-0	2923	DF-4	2876
	2	DF-4	2614	DF-9	4135	DF-4	2768	DF-0	1686
	3	DF-27	2192	DF-4	2896	DF-20	1885	DF-20	1651
	4	DF-13	1655	DF-28	2354	DF-26	1489	DF-26	1294
	5	DF-9	1620	DF-27	2018	DF-5	1459	DF-25	1204
	6	DF-28	1522	EB-170	1822	EB-206	1321	DF-24	1194
	7	EB-121	1460	DF-20	1621	DF-24	1267	DF-22	1164
	8	EB-63	1435	EB-124	1604	EB-175	1252	EB-274	1156
	9	EB-205	1398	DF-23	1557	EB-167	1220	EB-208	1132
	10	DF-25	1387	EB-307	1490	EB-321	1205	EB-59	1127

Data	**Rank**	**DenseNet201-coreml**		**DenseNet201-tflite**		**ResNet152-coreml**		**ResNet152-tflite**	
		Feature	Value	Feature	Value	Feature	Value	Feature	Value
CIFAR100	1	DF-86	4563	DF-181	3285	DF-202	2975	DF-91	4874
	2	DF-199	3141	DF-4	3133	DF-163	2974	DF-126	3802
	3	DF-138	3046	DF-184	2825	DF-0	2904	EB-227	3313
	4	DF-159	3012	DF-162	2748	DF-194	2773	DF-4	3107
	5	DF-91	2985	DF-80	2745	DF-122	2678	DF-72	3065
	6	DF-162	2972	DF-141	2734	EB-235	2551	EB-235	3047
	7	DF-4	2849	EB-251	2651	DF-179	2411	EB-233	2905
	8	EB-213	2812	EB-221	2634	DF-4	2351	EB-212	2839
	9	EB-232	2760	DF-125	2597	EB-257	2319	DF-61	2742
	10	DF-0	2749	EB-256	2577	EB-226	2150	DF-5	2633

Data	**Rank**	**LeNet1-coreml**		**LeNet1-tflite**		**LeNet5-coreml**		**LeNet5-tflite**	
		Feature	Value	Feature	Value	Feature	Value	Feature	Value
Fashion	1	DF-4	4083	DF-1	4760	DF-4	3685	DF-4	3526
	2	DF-0	3634	DF-4	4110	DF-8	2412	DF-28	1449
	3	DF-25	1943	DF-3	2202	EB-267	1843	EB-275	1417
	4	DF-28	1800	EB-32	1593	EB-258	1705	EB-138	1350
	5	EB-68	1785	EB-204	1585	DF-28	1650	EB-288	1326
	6	DF-5	1779	EB-283	1584	DF-5	1529	EB-55	1324
	7	EB-147	1593	DF-25	1518	EB-240	1326	DF-5	1308
	8	EB-297	1537	EB-262	1432	EB-174	1322	DF-23	1265
	9	EB-206	1531	EB-132	1427	EB-205	1304	EB-319	1262
	10	EB-132	1515	DF-22	1424	EB-133	1277	EB-311	1249

Data	**Rank**	**NIN-coreml**		**NIN-tflite**		**VGG19-coreml**		**VGG19-tflite**	
		Feature	Value	Feature	Value	Feature	Value	Feature	Value
Plant	1	DF-0	3335	DF-0	3826	DF-27	1736	DF-4	2834
	2	DF-4	3114	DF-4	2688	DF-4	1729	DF-52	2361
	3	DF-49	1994	DF-53	2632	EB-92	1676	EB-225	1974
	4	DF-76	1947	EB-97	2273	DF-52	1613	EB-256	1867
	5	DF-61	1804	DF-52	2142	DF-50	1582	DF-72	1733
	6	DF-5	1754	EB-88	2112	DF-74	1488	DF-57	1636
	7	EB-87	1748	EB-94	1975	DF-39	1433	DF-9	1587
	8	EB-94	1685	DF-49	1888	EB-247	1428	EB-323	1562
	9	DF-53	1679	DF-55	1784	EB-89	1350	DF-78	1438
	10	EB-148	1670	DF-50	1766	DF-20	1344	EB-264	1421

Data	**Rank**	**GRU-coreml**		**GRU-tflite**		**LSTM-coreml**		**LSTM-tflite**	
		Feature	Value	Feature	Value	Feature	Value	Feature	Value
News	1	DF-4	713	DF-4	791	DF-4	818	DF-4	673
	2	DF-24	377	DF-0	502	DF-11	458	DF-5	491
	3	DF-5	339	EB-87	329	DF-5	406	DF-16	349
	4	EB-72	315	EB-76	303	DF-16	383	DF-36	346
	5	DF-37	288	DF-5	298	EB-109	372	EB-83	327
	6	DF-46	272	DF-32	290	DF-43	371	EB-92	317
	7	DF-47	264	EB-120	282	EB-89	354	DF-12	309
	8	EB-60	246	DF-12	277	EB-92	294	EB-99	300
	9	DF-21	243	EB-98	266	DF-33	265	DF-11	295
	10	EB-148	239	DF-41	263	EB-85	251	DF-45	291

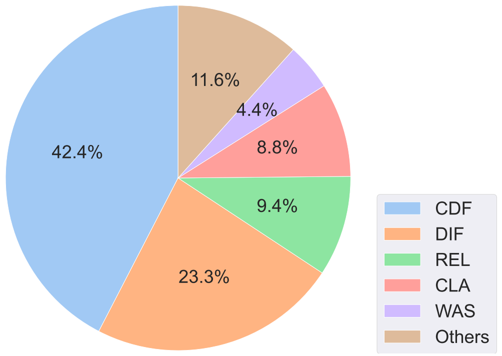

Fig. 3. Top five contributing features among all deviation features

The results of RQ5.2 are displayed in Table 18, where the scores signify the importance levels of individual features. For each pairing of model and dataset, we present the leading N features that contribute significantly. The abbreviations DF and EB denote deviation features and embedding features, respectively. The numerical values appended to the feature abbreviations indicate the corresponding feature indices. For instance, *DF-12* signifies the 12th deviation feature. From Table 18, we see that both DF and EB features consistently emerge among the foremost N contributors across diverse subjects. Overall, DF features exhibit a higher overall importance, constituting more than half of an equal share across each subject. Within the CIFAR10 dataset, in the top 10 features with the highest contributions, the proportion of DF features varies from 60% to 70%. In the context of the CIFAR100 dataset, DF features account for 60% to 80% within the top 10 contributing features. Within the News dataset, DF features cover a span of 60% to 70%.

The CDF features have the highest contributions to the effectiveness of PriCod compared to other deviation features. Figure 3 illustrates the specific contributions of each deviation feature to the effectiveness of PriCod. In particular, it represents the normalized results of the final importance values for each deviation feature. The detailed calculation process can be found in the experimental design of RQ 5.2. In the pie chart, if the proportion of a deviation feature is higher compared to others, it implies that this feature contributes more to the effectiveness of PriCod. Notably, CDF features have the highest total importance value, accounting for 42.4%. This suggests that CDF features, namely Coordinate Deviation Features, contribute the most to the effectiveness of PriCod. The second-highest proportion is attributed to DIF features, accounting for 23.3%.

Answer to RQ5.2: *Both deviation features and embedding features consistently demonstrate their presence among the top-N most influential attributes across a range of subjects. Notably, deviation features exhibit a greater overall importance. Among all the deviation features, the CDF features have the highest contributions to the effectiveness of PriCod compared to other deviation features.*

5.6 RQ6: Exploring whether uncertainty-based metrics can enhance the effectiveness of PriCod

Objectives: In the original PriCod, we generate embedding features for each test to indirectly reveal its proximity to the decision boundary. A prior study [99] suggests that uncertainty-based metrics can also reflect this proximity. Therefore, in this research question, we investigate whether integrating these metrics can enhance PriCod's effectiveness.

Experimental design: To assess whether the integration of uncertainty-based metrics can enhance the effectiveness of PriCod, we incorporate several uncertainty-based metrics into the original PriCod for the purpose of test prioritization. Specifically, we utilize six widely adopted uncertainty-based metrics [29, 96, 99], namely DeepGini, Vanilla SM, PCS, Entropy, Margin, and Least Confidence. The selection of these metrics is based on their widespread use in quantifying uncertainty in the context of DNN testing and their demonstrated effectiveness [39, 99]. The process of constructing the uncertainty feature vector for each test input $t \in T$ is outlined as follows:

- **Calculation of Confidence Scores:** Given a test input t, we compute its uncertainty scores using the aforementioned six uncertainty-based metrics.
- **Generation of Uncertainty Features:** For t, its uncertainty feature vector is generated by concatenating the uncertainty scores obtained from the six metrics. Consequently, for t, PriCod generated a final uncertainty feature vector, denoted as $[S_1, S_2, S_3, S_4, S_5, S_6]$, where each element S_i represents the uncertainty score calculated by the i_{th} uncertainty-based metric.

Finally, for the test t, we integrate its uncertainty features obtained above with embedding features and deviation features (the generation processes for these two types of features can be referred to in Section 3.2) to calculate the misclassification probability of this test. We represent this new PriCod method as PriCod$_u$. We compare the effectiveness of PriCod$_u$ and the original PriCod to determine whether integrating the uncertainty-based metrics can enhance PriCod's effectiveness.

Results: The experimental results for RQ6 are presented in Table 19. Here, PriCod$_u$ represents the variant of PriCod where uncertainty-based features are incorporated for test prioritization. Notably, for each case, we highlighted the approach with the highest effectiveness in gray. In Table 19, we see that the average effectiveness (measured by APFD) of the original PriCod slightly exceeds that of PriCod$_u$. Specifically, the average APFD for PriCod is 0.7810, while for PriCod$_u$, it is 0.7805, with a difference of only 0.0005. PriCod$_u$ demonstrates better effectiveness in a higher proportion of cases, accounting for 70% of all cases. However, in each individual case, the improvement of PriCod$_u$ relative to the original PriCod is slight. For instance, in the case of CIFAR10 with the AlexNet-coreml model, the APFD for PriCod is 0.721, while for PriCod$_u$, it is 0.722. The above experimental results indicate that uncertainty features do not enhance the performance of PriCod in terms of average results. In certain specific subjects, the inclusion of uncertainty features can lead to improvements, but the improvements are minor.

Answer to RQ6: *From the perspective of the average values, uncertainty features do not enhance the performance of PriCod. Although uncertainty features can lead to improvement in some specific subjects, the enhancement is minor.*

6 DISCUSSION

6.1 Limitations of PriCod

While PriCod has demonstrated its potential to improve test prioritization for compressed DNN models, it is crucial to recognize its limitations:

[*Model Compression Tools*] PriCod has currently undergone primary testing using two prevalent compression tools, TFLite and CoreML, which are widely utilized in the field. While our method has demonstrated its efficacy with models compressed using these tools, our future endeavors will involve assessing PriCod's performance across a broader spectrum of compression tools.

[*Model Compression Techniques*] Our approach primarily focuses on model compression through quantization, a prevalent method in the field of model compression. However, model compression encompasses a wide range of techniques beyond quantization. In the future, we intend to evaluate PriCod's effectiveness across a broader spectrum of compression methods. This broader assessment will be crucial in ensuring that PriCod remains versatile and capable of addressing various types of compressed DNN models.

Table 19. Effectiveness comparison among PriCod and PriCod$_u$ in terms of the APFD values on natural test inputs

Data	**Model**	**Approach**	
		PriCod	PriCod$_u$
CIFAR10	AlexNet-coreml	0.721	0.722
	AlexNet-tflite	0.720	0.721
	VGG16-coreml	0.781	0.783
	VGG16-tflite	0.781	0.782
CIFAR100	DenseNet201-coreml	0.788	0.790
	DenseNet201-tflite	0.796	0.789
	ResNet152-coreml	0.765	0.766
	ResNet152-tflite	0.786	0.778
Fashion	LeNet1-coreml	0.815	0.822
	LeNet1-tflite	0.815	0.821
	LeNet5-coreml	0.826	0.832
	LeNet5-tflite	0.824	0.829
Plant	NIN-coreml	0.795	0.798
	NIN-tflite	0.794	0.798
	VGG19-coreml	0.779	0.783
	VGG19-tflite	0.781	0.782
News	GRU-coreml	0.756	0.745
	GRU-tflite	0.757	0.744
	LSTM-coreml	0.771	0.763
	LSTM-tflite	0.770	0.762
	Average	0.7810	0.7805

[*Model Domains*] PriCod is primarily tailored for test prioritization in image-related domains. While image analysis is a typical application, the applicability of PriCod in other domains remains an area for exploration. In the future, we intend to evaluate PriCod's effectiveness in diverse domains beyond images.

6.2 Threats to Validity

Threats to Internal Validity. Internal validity threats primarily arise from the execution of PriCod and its variations, along with the test prioritization approaches being compared. To address this concern, we implement PriCod using the widely recognized TensorFlow library and the LightGBM framework. Regarding the implementation of the compared approaches, we utilize their original codebases as provided in their respective publications, aiming to minimize potential biases stemming from implementation. Another internal threat emerges from the inherent randomness associated with model training. To mitigate this concern and ensure the reliability of our experimental results, we conducted a statistical analysis by replicating the training procedure ten times. We presented the average experimental outcomes and calculated the statistical significance of the results.
Threats to External Validity. The primary external validity threats originate from the datasets and compressed DNN models employed in our study. To tackle these issues, we encompassed a diverse range of subjects, spanning natural, noisy, and adversarial data. As for the compressed DNN models, we utilized two widely adopted frameworks for model compression: TFLite and CoreML. Our objective is to perform a thorough evaluation of PriCod's effectiveness across various scenarios and to enhance the applicability of our conclusions.

7 RELATED WORK

7.1 Test prioritization for Deep Neural Networks

In the literature, a variety of techniques have been proposed to prioritize test inputs for Deep Neural Networks (DNNs)[29, 97, 99]. Feng *et al.* [29] introduced DeepGini [29], which identifies potentially misclassified tests by evaluating the model's confidence. This approach is built on the assumption that if a DNN assigns similar probabilities to each class for a test, it is more likely to be incorrectly predicted. Byun *et al.* [6] examined several metrics for prioritizing inputs that reveal software bugs, using white-box measurements of DNN sentiment. These metrics include softmax confidence (i.e., predicted probability for output categories in DNNs using softmax output layers), Bayesian uncertainty (i.e., uncertainty in prediction probability distributions for Bayesian Neural Networks), and input surprise (i.e., the disparity in neuron activation patterns between a test input and training data). Weiss *et al.* [99] investigated the effectiveness of different DNN test input prioritization methods, including notable confidence-based metrics like Vanilla Softmax, Prediction-Confidence Score (PCS), and Entropy. Wang et al.[97] introduced PRIMA, a technique for prioritizing test inputs for DNNs through intelligent mutation analysis. PRIMA enhances DNN test prioritization in two significant ways. Firstly, it can be applied not only to classification models but also to regression models. Secondly, PRIMA addresses scenarios where test inputs are generated using adversarial input generation methods[7], which could artificially increase the probability of an incorrect class assignment. However, PRIMA can not be adapted to compressed DNN models since the model mutation rules employed by PRIMA cannot be directly applied to compressed DNN models.

Zheng *et al.* [112] proposed CertPri, a DNN test input prioritization technique that focuses on measuring movement difficulty in the feature space. This method assesses the cost of moving test inputs closer to or farther from the class centers, providing a novel perspective on prioritization strategies. Al-Qadasi *et al.* [2] introduced a new metric, WFDR, for evaluating the effectiveness of prioritizing Dtest in the context of DNNs. The WFDR metric considers fault detection ratio and rate, incorporating adaptive weights to account for prioritization difficulty. This approach offers a comprehensive assessment framework for prioritization algorithms. Wei *et al.* [98] proposed EffiMAP, an efficient test case prioritization technique utilizing predictive mutation analysis. Without requiring a full mutation analysis, EffiMAP predicts the capability of test cases to expose model prediction failures based on information extracted from the test case execution trace. This pioneering work demonstrates the feasibility of predictive mutation analysis in ranking test cases for deep neural network testing. Tao *et al.* [88] introduced TPFL, a DNN test prioritization technique that employs dynamic spectrum analysis on each neuron. TPFL identifies suspicious neurons causing incorrect decisions in the DNN and prioritizes test inputs based on their ability to activate these neurons. This approach leverages key insights into the relationship between neuron activation and bug-revealing inputs.

Despite all the aforementioned methods being aimed at testing prioritization for DNNs, they primarily focus on traditional DNNs rather than compressed DNN models. Our proposed method, PriCod, is specifically designed for compressed DNN models. PriCod leverages the behavior disparities caused by model compression for test prioritization. These behavior disparities are a unique characteristic of compressed models. PriCod is the first to introduce this characteristic for test prioritization.

7.2 Test Prioritization for Traditional Software

In the realm of traditional software testing [3], the concept of test prioritization is a pivotal approach aimed at determining the most efficient sequence for executing test cases, enabling the swift identification of system defects [9, 23, 74, 83, 94, 106]. Rothermel *et al.* [83] pioneered introducing and comparing three distinct test case prioritization methods for regression testing, all rooted in test execution data. Their findings unequivocally showcased that each of the scrutinized prioritization methods substantially heightened the fault detection rate within the test suite. In the pursuit of evaluating the efficacy of diverse test case prioritization techniques in bug detection, Di Nardo *et al.* [23] conducted an empirical case study centered around coverage-based prioritization strategies applied to real-world regression faults. Similarly, Henard *et al.* [36] embarked on an extensive survey, undertaking a meticulous comparison of existing test prioritization approaches. Intriguingly, their investigation revealed marginal disparities between white-box [26, 27, 54, 106] and black-box strategies [35, 37, 52]. Another noteworthy advancement came from Chen *et al.* [9], who introduced the LET method, a pioneering approach to prioritizing test programs in the domain of compiler testing with the primary goal of enhancing efficiency. Chen *et al.* demonstrated the method's efficacy through two interconnected processes. The initial process involves learning, wherein the system identifies distinctive features of test programs and predicts the probability of a new test program uncovering bugs. Subsequently, the scheduling process comes into play, prioritizing test programs based on these predicted probabilities of bug discovery. This innovative dual-process framework proposed by Chen *et al.* constitutes a notable contribution to optimizing compiler testing strategies.

7.3 Deep Neural Network Testing

DeepXplore [79] is the first technique targeted at testing DNN models. It proposed neuron coverage, which measures the activation state of neurons, to guide the generation of test inputs. DeepXplore is based on differential testing, and it uses multiple models of a task to detect potential defects. To alleviate the need for multiple models under test, DeepTest [92] leverages metamorphic relations [13] that are expected to hold by a model as its test oracles. Both DeepXplore and DeepTest perturb their test inputs based on the gradient of deep learning models.

The preceding section has discussed test prioritization, which aims to reduce labeling costs and improve DNN testing efficiency. In addition to test prioritization, several test selection approaches have also been proposed to lower labeling costs. Test selection focuses on accurately estimating the accuracy of the entire dataset by labeling only a selected subset of test inputs. This approach effectively decreases the labeling costs associated with DNN testing. Li *et al.* [55] introduced CES (Cross Entropy-based Sampling), which performs test selection by minimizing the cross-entropy between the selected subset and the complete test set, ensuring that the distribution of the chosen test inputs resembles that of the original test set. Chen *et al.* [10] proposed PACE to guide DNN test selection. Initially, PACE clusters all test inputs into groups based on their testing characteristics. Then, PACE employs the MMD-critic algorithm [45] to select prototypes from each group. For test inputs not falling into any group, PACE utilizes adaptive random testing to select appropriate tests from them.

Wu *et al.* [101] introduced Stratified random Sampling with Optimum Allocation (SSOA), a framework that integrates sampling theory into the task of deep learning test input selection. SSOA leverages stratified random sampling and optimum allocation to provide an unbiased approach for selecting test inputs. This methodology contributes to mitigating biases in the test input selection process, enhancing the representativeness of the chosen inputs. Hao *et al.* [33] proposed Multiple-Objective Optimization-Based Test Input Selection (MOTS) as a method for selecting a more effective test subset to retrain DNN models. In contrast to existing approaches, MOTS considers both the uncertainty of test inputs and the diversity of the test subset. Employing the NSGA-II multiple-objective optimization algorithm, MOTS ensures that the selected test subset exhibits diverse features, providing enhanced support for DNN model retraining.

Wu *et al.* [100] introduced RNNtcs, a method for selecting test cases for Recurrent Neural Networks (RNNs) that combines clustering and uncertainty. RNNtcs aims to identify test cases that can effectively reveal RNN bugs by considering both the clustering structure and uncertainty. This approach is designed to reduce the cost of labeling by focusing on test cases with a higher likelihood of exposing model vulnerabilities. Liu *et al.* [58] proposed DeepState, a test suite selection tool tailored to the specific neural network structures of Recurrent Neural Network (RNN) models. DeepState reduces data labeling and computation costs by selecting data based on a stateful perspective of RNN. This perspective involves identifying potentially misclassified tests by capturing the state changes of neurons in RNN models. DeepState addresses the unique challenges posed by RNN structures in the context of test input selection.

7.4 Test Generation approaches for Compressed DNN models

In the literature, several test generation approaches have been proposed for compressed DNN models. Odena and Goodfellow introduced TensorFuzz [73], a pioneering method that utilized coverage-guided fuzzing as a test generation approach. TensorFuzz aimed to reveal difference-inducing inputs between a well-trained DNN and its quantized counterpart. By employing a coverage-guided strategy, TensorFuzz efficiently explored the input space, exposing discrepancies in the behavior of compressed DNN models.

Yahmed *et al.* [105] proposed DiverGet, a search-based testing framework designed specifically for quantization assessment in compressed DNN models. DiverGet introduces a structured space of metamorphic relations that simulate natural distortions on input data. These metamorphic relations are then systematically explored to optimize the revelation of disagreements among DNNs subjected to different arithmetic precision. By defining and strategically navigating this metamorphic space, DiverGet provides a comprehensive approach to evaluating the impact of quantization on DNN models.

Xie *et al.* [103] proposed DiffChaser, a novel automated black-box disagreement detection technique tailored for multiple variants of a DNN. The core premise behind DiffChaser is the identification of similarities in decision boundaries between a DNN and its quantization-aware training (QC) version variants. The rationale is that the decision boundaries tend to exhibit resemblance, particularly in proximity to the boundary itself. Consequently, inputs near these decision boundaries are more likely to capture the discrepancies in decision boundaries, representing the disagreement among the DNN models. DiffChaser leverages prediction uncertainty as a guiding metric and automatically generates inputs that lie in the vicinity of decision boundaries to unveil the distinctions between DNN variants.

8 CONCLUSION

To address the challenge of labeling-cost reduction in the context of testing compressed DNN models, we proposed PriCod, a novel test prioritization approach designed to identify and prioritize potentially misclassified tests. PriCod is rooted in two fundamental premises: firstly, that significant prediction deviations between compressed and original DNN models signify a greater likelihood of test input misclassification, and secondly, that test inputs situated near decision boundaries are more susceptible to misclassification. Building upon these premises, PriCod generates two distinct feature types for each test input for the purpose of test prioritization: deviation features, quantifying the prediction deviation caused by model compression, and embedding features, which indirectly reflect proximity to decision boundaries by leveraging intrinsic information about test inputs. These features are combined to calculate the misclassification probability of each test input. Subsequently, PriCod ranks all tests in descending order based on their misclassification probability. We conducted a comprehensive assessment of PriCod's performance, using different types of test inputs and various test prioritization techniques. Our findings consistently showcased PriCod's superior performance, revealing an average improvement from 7.43% to 55.89% for natural test inputs, 7.92% to 52.91% for noisy inputs, and 7.03% to 51.59% for adversarial inputs compared to existing methods.

Availability. All artifacts are available in the following public repository:

https://github.com/yinghuali/PriCod

ACKNOWLEDGEMENTS

This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 949014).

REFERENCES

[1] Zohreh Aghababaeyan, Manel Abdellatif, Lionel Briand, S Ramesh, and Mojtaba Bagherzadeh. 2023. Black-box testing of deep neural networks through test case diversity. *IEEE Transactions on Software Engineering* (2023).

[2] Hamzah Al-Qadasi, Yliès Falcone, and Saddek Bensalem. 2023. Difficulty and severity-oriented metrics for test prioritization in deep learning systems. In *2023 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 40–48.

[3] Nadia Alshahwan, Mark Harman, and Alexandru Marginean. 2023. Software Testing Research Challenges: An Industrial Perspective. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10.

[4] Zeynep Batmaz, Ali Yurekli, Alper Bilge, and Cihan Kaleli. 2019. A review on deep learning for recommender systems: challenges and remedies. *Artificial Intelligence Review* 52 (2019), 1–37.

[5] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. 2017. Adversarial patch. *arXiv preprint arXiv:1712.09665* (2017).

[6] Taejoon Byun, Vaibhav Sharma, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. 2019. Input prioritization for testing neural networks. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 63–70.

[7] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*. Ieee, 39–57.

[8] Venkat Chandrasekaran and Parikshit Shah. 2017. Relative entropy optimization and its applications. *Mathematical Programming* 161 (2017), 1–32.

[9] Junjie Chen, Guancheng Wang, Dan Hao, Yingfei Xiong, Hongyu Zhang, Lu Zhang, and Bing Xie. 2018. Coverage prediction for accelerating compiler testing. *IEEE Transactions on Software Engineering* 47, 2 (2018), 261–278.

[10] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. 2020. Practical accuracy estimation for efficient deep neural network testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 4 (2020), 1–35.

[11] Qi Chen, Sihai Tang, Qing Yang, and Song Fu. 2019. Cooper: Cooperative perception for connected autonomous vehicles based on 3d point clouds. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 514–524.

[12] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[13] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 2020. Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543* (2020).

[14] Wei-Yu Chiu, Gary G Yen, and Teng-Kuei Juan. 2016. Minimum manhattan distance approach to multiple criteria decision making in multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation* 20, 6 (2016), 972–985.

[15] Tejalal Choudhary, Vipul Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review* 53 (2020), 5113–5155.

[16] Israel Cohen, Yiteng Huang, Jingdong Chen, Jacob Benesty, Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. *Noise reduction in speech processing* (2009), 1–4.

[17] Xueqi Dang, Yinghua Li, Mike Papadakis, Jacques Klein, Tegawendé F Bissyandé, and Yves LE Traon. 2023. GraphPrior: Mutation-based Test Input Prioritization for Graph Neural Networks. *ACM Transactions on Software Engineering and Methodology* (2023).

[18] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems* 3 (2021), 800–811.

[19] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. 2021. Tensorflow lite micro: Embedded machine learning for tinyml systems. *Proceedings of Machine Learning and Systems* 3 (2021), 800–811.

[20] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[22] Rahul Dey and Fathi M Salem. 2017. Gate-variants of gated recurrent unit (GRU) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 1597–1600.

[23] Daniel Di Nardo, Nadia Alshahwan, Lionel Briand, and Yvan Labiche. 2013. Coverage-based test case prioritisation: An industrial case study. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, 302–311.

[24] Len Du. 2020. How much deep learning does neural style transfer really need? an ablation study. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3150–3159.

[25] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering* 28, 2 (2002), 159–182.

[26] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 235–245.

[27] Emelie Engström, Per Runeson, and Mats Skoglund. 2010. A systematic review on regression test selection techniques. *Information and Software Technology* 52, 1 (2010), 14–30.

[28] Umi Fadlilah, Bana Handaga, et al. 2021. The development of android for Indonesian sign language using tensorflow lite and CNN: an initial study. In *Journal of Physics: Conference Series*, Vol. 1858. IOP Publishing, 012085.

[29] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 177–188.

[30] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive test selection for deep neural networks. In *Proceedings of the 44th International Conference on Software Engineering*. 73–85.

[31] Víctor González-Castro, Rocío Alaiz-Rodríguez, and Enrique Alegre. 2013. Class distribution estimation based on the Hellinger distance. *Information Sciences* 218 (2013), 146–164.

[32] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[33] Yao Hao, Zhiqiu Huang, Hongjing Guo, and Guohua Shen. 2023. Test Input Selection for Deep Neural Network Enhancement Based on Multiple-Objective Optimization. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 534–545.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[35] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. 2013. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22, 1 (2013), 1–42.

[36] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 523–534.

[37] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. 2014. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering* 40, 7 (2014), 650–670.

[38] Han Hu, Yujin Huang, Qiuyuan Chen, Terry Yue Zhuo, and Chunyang Chen. 2023. A First Look at On-device Models in iOS Apps. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–30.

[39] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Wei Ma, Mike Papadakis, and Yves Le Traon. 2021. Towards Exploring the Limitations of Active Learning: An Empirical Study. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 917–929.

[40] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[41] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. 2018. The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 954–960.

[42] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. 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*. 67–78.

[43] Gunel Jahangirova and Paolo Tonella. 2020. An empirical evaluation of mutation operators for deep learning systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 74–84.

[44] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[45] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. 2016. Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems* 29 (2016).

[46] Tae Kyun Kim. 2015. T test as a parametric statistic. *Korean journal of anesthesiology* 68, 6 (2015), 540–546.

[47] Torleiv Klove, Te-Tsung Lin, Shi-Chun Tsai, and Wen-Guey Tzeng. 2010. Permutation arrays under the Chebyshev distance. *IEEE Transactions on Information Theory* 56, 6 (2010), 2611–2617.

[48] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[50] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. 2018. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. Chapman and Hall/CRC, 99–112.

[51] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[52] Yves Ledru, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. 2012. Prioritizing test cases with string distances. *Automated Software Engineering* 19, 1 (2012), 65–95.
[53] Yinghua Li, Xueqi Dang, Haoye Tian, Tiezhu Sun, Zhijie Wang, Lei Ma, Jacques Klein, and Tegawende F Bissyande. 2022. AI-driven Mobile Apps: an Explorative Study. *arXiv preprint arXiv:2212.01635* (2022).
[54] Zheng Li, Mark Harman, and Robert M Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering* 33, 4 (2007), 225–237.
[55] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. 2019. Boosting operational dnn testing efficiency through conditioning. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 499–509.
[56] Leo Liberti, Carlile Lavor, Nelson Maculan, and Antonio Mucherino. 2014. Euclidean distance geometry and applications. *SIAM review* 56, 1 (2014), 3–69.
[57] Min Lin, Qiang Chen, and Shuicheng Yan. 2013. Network in network. *arXiv preprint arXiv:1312.4400* (2013).
[58] Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen. 2022. DeepState: Selecting Test Suites to Enhance the Robustness of Recurrent Neural Networks. In 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). 598–609. *Google Scholar Google Scholar Digital Library Digital Library* (2022).
[59] Yuanfei Luo, Mengshuo Wang, Hao Zhou, Quanming Yao, Wei-Wei Tu, Yuqiang Chen, Wenyuan Dai, and Qiang Yang. 2019. Autocross: Automatic feature crossing for tabular data in real-world applications. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1936–1945.
[60] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.
[61] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 100–111.
[62] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. 2021. Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 2 (2021), 1–22.
[63] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).
[64] Nattaya Mairittha, Tittaya Mairittha, and Sozo Inoue. 2019. On-device deep learning inference for efficient activity data collection. *Sensors* 19, 15 (2019), 3434.
[65] Nattaya Mairittha, Tittaya Mairittha, and Sozo Inoue. 2020. Improving activity data collection with on-device personalization using fine-tuning. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*. 255–260.
[66] Nattaya Mairittha, Tittaya Mairittha, and Sozo Inoue. 2020. On-device deep personalization for robust activity data collection. *Sensors* 21, 1 (2020), 41.
[67] MD Malkauthekar. 2013. Analysis of Euclidean distance and Manhattan distance measure in Face recognition. In *Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)*. IET, 503–507.
[68] Tong Meng, Xuyang Jing, Zheng Yan, and Witold Pedrycz. 2020. A survey on machine learning for data fusion. *Information Fusion* 57 (2020), 115–129.
[69] Agnieszka Mikołajczyk and Michał Grochowski. 2018. Data augmentation for improving deep learning in image classification problem. In *2018 international interdisciplinary PhD workshop (IIPhDW)*. IEEE, 117–122.
[70] Sharada P Mohanty, David P Hughes, and Marcel Salathé. 2016. Using deep learning for image-based plant disease detection. *Frontiers in plant science* 7 (2016), 1419.
[71] Hieu V Nguyen and Li Bai. 2010. Cosine similarity metric learning for face verification. In *Asian conference on computer vision*. Springer, 709–720.
[72] Quang Hung Nguyen, Hai-Bang Ly, Lanh Si Ho, Nadhir Al-Ansari, Hiep Van Le, Van Quan Tran, Indra Prakash, and Binh Thai Pham. 2021. Influence of data splitting on performance of machine learning models in prediction of shear strength of soil. *Mathematical Problems in Engineering* 2021 (2021), 1–15.
[73] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. PMLR, 4901–4911.
[74] Rongqi Pan, Mojtaba Bagherzadeh, Taher A Ghaleb, and Lionel Briand. 2022. Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering* 27, 2 (2022), 29.

[75] Victor M Panaretos and Yoav Zemel. 2019. Statistical aspects of Wasserstein distances. *Annual review of statistics and its application* 6 (2019), 405–431.

[76] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2017. Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. *IEEE Transactions on Software Engineering* 44, 2 (2017), 122–158.

[77] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2018. A large scale empirical comparison of state-of-the-art search-based test case generators. *Information and Software Technology* 104 (2018), 236–256.

[78] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[79] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[80] Luis Perez and Jason Wang. 2017. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621* (2017).

[81] Friedrich Pillichshammer. 2000. On the sum of squared distances in the Euclidean plane. *Archiv der Mathematik* 74, 6 (2000), 472–480.

[82] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).

[83] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[84] Neha Sharma, Vibhor Jain, and Anju Mishra. 2018. An analysis of convolutional neural networks for image classification. *Procedia computer science* 132 (2018), 377–384.

[85] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of big data* 6, 1 (2019), 1–48.

[86] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[87] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. 2021. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *30th USENIX Security Symposium (USENIX Security 21)*. 1955–1972.

[88] Yali Tao, Chuanqi Tao, Hongjing Guo, and Bohan Li. 2022. TPFL: Test Input Prioritization for Deep Neural Networks Based on Fault Localization. In *International Conference on Advanced Data Mining and Applications*. Springer, 368–383.

[89] Luke Taylor and Geoff Nitschke. 2018. Improving deep learning with generic data augmentation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1542–1547.

[90] Mohit Thakkar. 2019. *Beginning machine learning in ios: CoreML framework*. Apress.

[91] Mohit Thakkar and Mohit Thakkar. 2019. Introduction to core ML framework. *Beginning Machine Learning in iOS: CoreML Framework* (2019), 15–49.

[92] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*. 303–314.

[93] Yongqiang Tian, Wuqi Zhang, Ming Wen, Shing-Chi Cheung, Chengnian Sun, Shiqing Ma, and Yu Jiang. 2023. Finding Deviated Behaviors of the Compressed DNN Models for Image Classifications. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–32.

[94] Paolo Tonella, Paolo Avesani, and Angelo Susi. 2006. Using the case-based ranking methodology for test case prioritization. In *2006 22nd IEEE international conference on software maintenance*. IEEE, 123–133.

[95] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. 2019. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1588–1597.

[96] Dan Wang and Yi Shang. 2014. A new active labeling method for deep learning. In *2014 International joint conference on neural networks (IJCNN)*. IEEE, 112–119.

[97] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing test inputs for deep neural networks via mutation analysis. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 397–409.

[98] Zhengyuan Wei, Haipeng Wang, Imran Ashraf, and WK Chan. 2022. Predictive Mutation Analysis of Test Case Prioritization for Deep Neural Networks. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 682–693.

[99] Michael Weiss and Paolo Tonella. 2022. 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*. 139–150.

[100] Xiaoxue Wu, Jinjin Shen, Wei Zheng, Lidan Lin, Yulei Sui, and Abubakar Omari Abdallah Semasaba. 2023. RNNtcs: A test case selection method for Recurrent Neural Networks. *Knowledge-Based Systems* 279 (2023), 110955.

[101] Zhuo Wu, Zan Wang, Junjie Chen, Hanmo You, Ming Yan, and Lanjun Wang. 2024. Stratified random sampling for neural network test input selection. *Information and Software Technology* 165 (2024), 107331.

[102] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[103] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. 2019. Diffchaser: Detecting disagreements for deep neural networks. International Joint Conferences on Artificial Intelligence Organization.

[104] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*. 2125–2136.

[105] Ahmed Haj Yahmed, Houssem Ben Braiek, Foutse Khomh, Sonia Bouzidi, and Rania Zaatour. 2022. Diverget: a search-based software testing approach for deep neural network quantization assessment. *Empirical Software Engineering* 27, 7 (2022), 193.

[106] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120.

[107] Shin Yoo, Mark Harman, Paolo Tonella, and Angelo Susi. 2009. Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *Proceedings of the eighteenth international symposium on Software testing and analysis*. 201–212.

[108] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation* 31, 7 (2019), 1235–1270.

[109] Xiangyu Yue, Bichen Wu, Sanjit A Seshia, Kurt Keutzer, and Alberto L Sangiovanni-Vincentelli. 2018. A lidar point cloud generator: from a virtual world to autonomous driving. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. 458–464.

[110] Zeroshot. 2023. Twitter Financial News Topic Dataset. (2023). https://huggingface.co/datasets/zeroshot/twitter-financial-news-topic

[111] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36.

[112] Haibin Zheng, Jinyin Chen, and Haibo Jin. 2023. CertPri: Certifiable Prioritization for Deep Neural Networks via Movement Cost in Feature Space. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1–13.

[113] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).