Test Input Prioritization for 3D Point Clouds

YINGHUA LI, University of Luxembourg, Luxembourg
XUEQI DANG*, University of Luxembourg, Luxembourg
LEI MA, The University of Tokyo, Tokyo, Japan
JACQUES KLEIN, University of Luxembourg, Luxembourg
YVES LE TRAON, University of Luxembourg, Luxembourg
TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg

Three-dimensional (3D) point cloud applications have become increasingly prevalent in diverse domains, showcasing their efficacy in various software systems. However, testing such applications presents unique challenges due to the high-dimensional nature of 3D point cloud data and the vast number of possible test cases. Test input prioritization has emerged as a promising approach to enhance testing efficiency by prioritizing potentially misclassified test cases during the early stages of the testing process. Consequently, this enables the early labeling of critical inputs, leading to a reduction in the overall labeling cost. However, applying existing prioritization methods to 3D point cloud data is constrained by several factors: 1) Inadequate consideration of crucial spatial information, and 2) susceptibility to noises inherent in 3D point cloud data. In this paper, we propose PCPrior, the first test prioritization approach specifically designed for 3D point cloud test cases. The fundamental concept behind PCPrior is that test inputs closer to the decision boundary of the model are more likely to be predicted incorrectly. To capture the spatial relationship between a point cloud test and the decision boundary, we propose transforming each test (a point cloud) into a low-dimensional feature vector, towards indirectly revealing the underlying proximity between a test and the decision boundary. To achieve this, we carefully design a group of feature generation strategies, and for each test input, we generate four distinct types of features, namely, spatial features, mutation features, prediction features, and uncertainty features. Through a concatenation of the four feature types, PCPrior assembles a final feature vector for each test. Subsequently, a ranking model is employed to estimate the probability of misclassification for each test based on its feature vector. Finally, PCPrior ranks all tests based on their misclassification probabilities. We conducted an extensive study based on 165 subjects to evaluate the performance of PCPrior, encompassing both natural and noisy datasets. The results demonstrate that PCPrior outperforms all the compared test prioritization approaches, with an average improvement of 10.99%~66.94% on natural datasets and 16.62%~53% on noisy datasets.

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 advent of point cloud data has revolutionized various fields, such as computer vision [15, 84], autonomous driving [36, 96], augmented reality [8, 59, 60] and smart cities [63, 63], by enabling highly accurate and detailed

*Corresponding author.

Authors' addresses: Yinghua Li, yinghua.li@uni.lu, University of Luxembourg, Luxembourg; Xueqi Dang, xueqi.dang@uni.lu, University of Luxembourg, Luxembourg; Lei Ma, ma.lei@acm.org, The University of Tokyo, Tokyo, Japan; 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 1049-331X/2024/1-ART
https://doi.org/10.1145/3643676

representation of real-world environments. A Point cloud [92] refers to a collection of three-dimensional data points in space, typically representing the surface geometry or shape of real-world objects or environments. Each data point in a point cloud is defined by its spatial coordinates (x, y, z) and, in some cases, additional attributes such as color or intensity values. Figure 1 illustrates an example of a point cloud representing the shape of a car, composed of thousands of individual points. To showcase the inherent three-dimensional attributes of the point cloud, we present multiple perspectives of the object from different viewing angles. From specific angles, the car is easily identifiable and recognizable. However, from some angles, it becomes challenging to identify the object as a car. Point clouds are commonly generated using various sensing technologies, including LiDAR (Light Detection and Ranging) [24], depth cameras [18], or structured light scanners [77], which capture the physical measurements of points in the environment.

Compared to two-dimensional data like images, 3D point clouds have inherent differences and significant advantages. First, 3D point clouds offer a three-dimensional depiction of objects, resulting in higher accuracy and reliability when identifying complex 3D shapes and volumes. Moreover, point cloud data can directly capture surface details and morphology of objects, making them difficult to be replaced by images in many practical applications. Consequently, the integration of point cloud processing in safety-critical applications, such as autonomous driving [19, 96], medical imaging [87], and industrial automation [86], has become increasingly prevalent. For instance, 3D point clouds can be utilized for autonomous driving in the context of obstacle detection and perception [80]. More specifically, 3D point cloud data obtained from LiDAR (Light Detection and Ranging) sensors [24] can provide a rich and detailed representation of the surrounding environment in three-dimensional space. By leveraging this data, it becomes feasible to identify and localize various objects on the road, such as automobiles, pedestrians, cyclists, and obstacles. Leveraging these 3D data, autonomous driving systems can employ 3D classification models to detect and categorize objects, thereby guiding the avoidance of obstacles. Hence, the accuracy of these 3D classification models plays a pivotal role in ensuring the safety of autonomous driving.

In recent years, Deep Neural Networks (DNNs) have emerged as a powerful tool for various computer vision tasks [38, 43], and their application to 3D point cloud data has garnered significant attention. Ensuring the reliability of DNNs operating on point cloud data is crucial for safe and efficient functioning. DNN testing [39, 90] has become a widely adopted approach to assess and ensure the quality of such networks. Nevertheless, prior investigations [14, 28, 89] have highlighted a central challenge pertaining to DNN testing: the significant cost incurred in labeling test inputs to verify the accuracy of DNN predictions. First, the scale of the test set is typically extensive. Second, manual labelling is mainstream, typically necessitating the involvement of multiple annotators to ensure the accuracy and consistency of the labeling process for each test input.

The challenges are further compounded in the case of 3D point cloud data. In addition to the aforementioned obstacles, labeling point cloud data presents additional distinctive challenges compared to traditional image/text data.

- **Data representation** Image data is represented as two-dimensional matrices, with each pixel having a distinct position and value. In contrast, point cloud data comprises an unordered set of points, each possessing three-dimensional coordinates and additional attributes such as color and normals. This distinctive data representation significantly increases the complexity of labeling, necessitating additional processing and interpretation steps.
- **Sparsity of point clouds** Point cloud data is generally characterized by sparsity compared to image data. There can be missing points or noise in the point cloud, and the distribution of points is non-uniform. This inherent sparsity poses challenges for accurate labeling.
- **Expert knowledge for 3D point clouds** Labeling 3D point cloud data necessitates domain-specific expertise due to its unique characteristics. With a large number of three-dimensional points, each with its own coordinates and potential attributes, accurately labeling 3D point cloud data requires expert knowledge. This

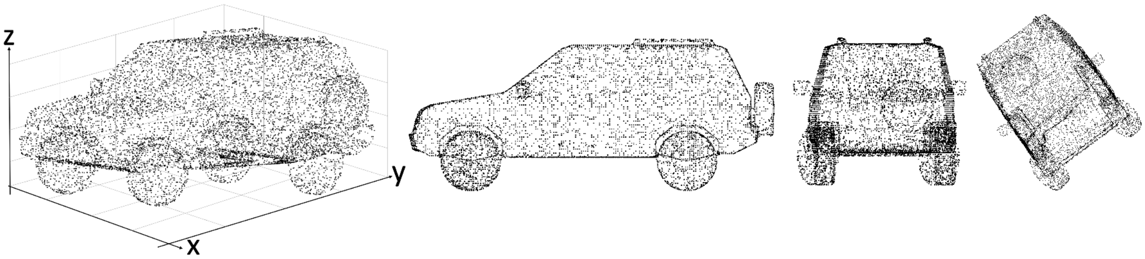


Fig. 1. Example of Point cloud test cases

expertise is crucial for understanding and interpreting the geometric attributes, shapes, and potentially semantic information conveyed by the points.

To address the issue of labeling cost in the context of DNNs, previous research efforts [28] have primarily focused on test prioritization, which aims to prioritize test inputs that are more likely to be misclassified by the model. By allocating resources to label these challenging inputs first, developers can ensure priority for critical test cases, ultimately resulting in reduced overall labeling costs. Existing test prioritization approaches [28, 89, 90] can be broadly categorized into two main groups: coverage-based and confidence-based. Coverage-based techniques prioritize test inputs based on the coverage of neurons [53, 94]. In contrast, confidence-based approaches operate under the assumption that test inputs for which the model exhibits lower confidence are more likely to be misclassified. Notably, confidence-based approaches have been demonstrated to be more effective than coverage-based approaches in the existing studies [28]. Weiss *et al.* [90] conducted a comprehensive exploration of diverse test input prioritization techniques, encompassing several confidence-based metrics that can be adapted to 3D point cloud data, such as DeepGini, Vanilla Softmax, Prediction-Confidence Score (PCS), and Entropy.

Although the confidence-based test prioritization approaches have demonstrated efficacy in specific contexts such as image and text data, they encounter several limitations when applied to 3D point cloud data.

- **Noises in 3D point cloud data** 3D point cloud data can exhibit inherent noise, which arises from various sources such as sensor noise and non-uniform sampling density. These noise factors can affect the effectiveness of confidence-based approaches. Specifically, in the presence of noise, the model can erroneously assign a high probability to an incorrect category for a given test sample. Consequently, confidence-based approaches incorrectly assume that the model is highly confident of this particular test, considering it will not be misclassified. However, the model's prediction on this test sample is indeed incorrect (i.e., this test is misclassified by the model).
- **Missing crucial spatial features** Confidence-based methods typically rely on the model's prediction confidence on test samples. However, in the case of 3D point cloud data, the point cloud exhibits complex spatial characteristics, and relying solely on the confidence feature of the model's prediction for test prioritization is limited. In other words, confidence-based methods fail to fully leverage the informative features inherent in point cloud data for test prioritization.

In addition to coverage-based and confidence-based techniques, Wang *et al.* [89] proposed PRIMA, which leverages mutation analysis and learning-to-rank methodologies for test input prioritization in DNNs. However, although demonstrating effectiveness in the domain of DNN test prioritization, PRIMA faces challenges when applied to 3D point cloud data. The reason is that: 1) the mutation operators utilized in PRIMA are primarily designed for two-dimensional images, text, and predefined features. These operators are not directly applicable to 3D point cloud data. In contrast to conventional image or text data, 3D point clouds exhibit a distinctive three-dimensional representation characterized by a substantial quantity of points; 2) even when considering the possibility of utilizing dimensionality reduction techniques to transform 3D data into two-dimensional images and integrating them into PRIMA, practical issues emerge. The execution flow of PRIMA necessitates the mutated two-dimensional images to be fed into the evaluated model for comparing the prediction results between mutants and original inputs. However, the model employed for 3D point clouds is inherently tailored to process three-dimensional data and lacks the capability to classify the mutated two-dimensional images. As a result, even in scenarios where dimensionality reduction tools are accessible, PRIMA remains unsuitable for accommodating 3D point cloud data.

In this paper, we propose PCPrior (3D **P**oint **C**loud Test **Prior**itization), a novel test prioritization approach specifically designed for 3D point cloud test cases. PCPrior leverages the unique characteristics of 3D point clouds to prioritize tests. It is crucial to emphasize that our approach focuses on datasets where each 3D point cloud corresponds to an individual test case. Therefore, each test case is constituted by a collection of points. The core idea behind the PCPrior framework is that: test inputs situated closer to the decision boundary of the model are more likely to be predicted incorrectly, which has been proven in the prior research [57]. PCPrior aims to prioritize such possibly-misclassified tests higher.

To reflect the distance between a test (a point cloud) and the decision boundary, we adopt a vectorization approach to map each test to a low-dimensional space, indirectly revealing the proximity between the point cloud data and the decision boundary. Based on this vectorization strategy, we design a diverse set of features to characterize a point cloud test, including Spatial Features (SF), Mutation Features (MF), Prediction Features (PF), and Uncertainty Features (UF). Notably, SF and MF are specifically designed based on the characteristics of point clouds. Specifically, these features play a pivotal role in capturing essential aspects, including the spatial properties of the point cloud, mutation information present in the input, predictions generated by the DNN model, and the corresponding confidence levels. PCPrior constructs a comprehensive feature vector through the concatenation of these four feature types and leverages a ranking model to learn from it for effective test prioritization.

Compared to existing test prioritization approaches, PCPrior has the following advantages:

- **Tailored for 3D Point Cloud Data** PCPrior is specifically designed to address the challenges of test prioritization for 3D point cloud data. Unlike existing approaches that focus on 2D images or text data, PCPrior leverages the distinctive characteristics of 3D point clouds and provides a more targeted approach for prioritizing tests.
- **Effective Utilization of Spatial Features** PCPrior leverages the spatial features of 3D point clouds, which are essential for understanding the geometric attributes and shapes of objects in the data. Unlike confidence-based approaches that solely rely on prediction confidence, PCPrior incorporates spatial features into the prioritization process. By considering the spatial properties of the point cloud data, PCPrior can effectively capture the informative features necessary for accurate test prioritization.
- **Comprehensive Feature Generation Mechanism** In addition to incorporating spatial characteristics, PCPrior integrates confidence-based features while also taking into account mutation and prediction features. By combining these features into a comprehensive feature vector, PCPrior captures a rich set of information that enhances the effectiveness of test prioritization.

PCPrior exhibits broad applicability across diverse domains. As a case in point, in the field of autonomous driving, when testing a 3D shape classification model, the utilization of sensors facilitates the collection of unlabeled test sets comprising surrounding 3D point clouds. PCPrior can be utilized to identify and prioritize test instances that are more likely to be misclassified by the model. By focusing on labeling these possibly-misclassified test inputs, it results in a reduction of both labeling time and the manual efforts involved in the labeling process.

To evaluate the effectiveness of PCPrior, we conduct an extensive experimental evaluation on a diverse set of 165 subjects, encompassing both natural datasets and noisy datasets. We compare PCPrior with several existing test prioritization approaches that have demonstrated effectiveness in prior studies [28, 90]. The evaluation metrics include the Average Percentage of Fault-Detection (APFD) [94] and Percentage of Fault Detected (PFD) [28], which are standard and widely-adopted metrics for test prioritization. The experimental results demonstrate the superiority of PCPrior over existing test prioritization techniques. Specifically, when applied to natural datasets, PCPrior consistently outperforms all the comparative test prioritization approaches, yielding an improvement ranging from 10.99% to 66.94% in terms of APFD. Moreover, on noisy datasets, the improvement ranges from 16.62% to 53%. We publish our dataset, results, and tools to the community on Github[1].

Our work has the following major contributions:

- **Approach** We propose PCPrior, the first test prioritization approach specifically for 3D point cloud data. To this end, we design four types of features that can comprehensively extract information from a 3D point cloud test. We employ effective ranking models to learn from the generated features for test prioritization.
- **Study** We conduct an extensive study based on 165 3D point cloud subjects involving natural and noisy datasets. We compare PCPrior with multiple test prioritization approaches. Our experimental results demonstrate the effectiveness of PCPrior.
- **Performance Analysis** We compare the contributions of different types of features to the effectiveness of PCPrior. We also investigate the impact of main parameters in PCPrior.

2 BACKGROUND

2.1 Deep Learning for 3D Point Clouds

The rapid advancements in sensor technologies, such as LiDAR (Light Detection and Ranging) [24] and RGB-D (Red-Green-Blue Depth) cameras [49], have led to the proliferation of three-dimensional (3D) point cloud data. These representations find significant utility in various fields, including medical treatment [97], autonomous driving [16, 19], and robotics [70, 98]. Typically, a point cloud represents a collection of data points in 3D space, each point typically denoted by its spatial coordinates (x, y, z) and, in some cases, additional attributes like color or intensity values [1]. Figure 1 illustrates one concrete example of a point cloud, showcasing the shape of a car. Each point cloud comprises numerous individual points.

The emergence of Deep Learning [6, 50], particularly Convolutional Neural Networks (CNNs) and PointNet [72], has revolutionized the analysis and understanding of 3D point cloud data. Moreover, the availability of numerous publicly accessible datasets, such as ModelNet [93], ShapeNet [10], and S3DIS [4], has played a pivotal role in stimulating research endeavors focused on deep learning techniques applied to 3D point clouds. This surge in research has led to the development of numerous methods addressing various problems in point cloud processing. One extensively studied problem in this domain is **three-dimension (3D) shape classification**, which focuses on utilizing DNNs to classify three-dimensional shapes. For example, in the field of autonomous driving, 3D shape classification can be utilized to categorize various objects on the road, such as vehicles, pedestrians, traffic signs, etc. By accurately classifying these objects, the autonomous driving system can better understand the surrounding environment, enabling more precise decision-making.

3D shape classification typically involves three main steps: **1) Learning individual point embeddings** Initially, each point in the point cloud undergoes processing to acquire its embedding representation. **2) Obtaining global shape embedding** Subsequently, these individual point embeddings are aggregated to generate the global shape embedding for the entire point cloud. This step aims to capture the overall structure and shape characteristics of the entire point cloud. **3) Classification Processing** Finally, the global shape embedding is input into several fully connected layers for classification. These layers are responsible for determining the category of the 3D shape represented by the point cloud based on the extracted global features.

In the literature [72, 73, 88, 92], several approaches have been proposed to tackle the challenge of 3D shape classification, such as PointConv [92], Dynamic Graph Convolutional Neural Network (DGCNN) [88], and PointNet [72]. PointConv is a specialized convolutional neural network designed for processing 3D point clouds. Training multi-layer perceptrons on local point coordinates enables the construction of deep networks directly

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

on 3D point clouds for efficient analysis. DGCNN, tailored for 3D point cloud data, leverages intrinsic spatial relationships by modeling them as graphs. Through graph convolutions and dynamic adaptation of the graph structure based on input data, DGCNN effectively learns and processes point cloud representations. PointNet, a widely adopted architecture for 3D point cloud data, incorporates a shared multi-layer perceptron (MLP) with max-pooling for local feature extraction and a symmetric function for aggregating global features. T-Net layers enable PointNet to learn transformation matrices, enhancing its robustness to input variations. PointNet has demonstrated impressive capabilities in 3D shape classification.

2.2 Mutation Testing

Mutation testing in traditional software engineering In the field of software testing [64, 65, 95], mutation testing [39, 67] presents a robust methodology for evaluating the effectiveness of a test suite in identifying code defects. The primary objective of mutation testing revolves around assessing the test suite's capacity to detect and localize faults within the code. The fundamental premise is that: if a test case can successfully uncover a mutation, thereby revealing a discrepancy in program behavior compared to the original code, it signifies the test case's potential to identify bugs in real-world scenarios. These mutations are intentionally introduced into the original program through simple syntactic modifications, resulting in the creation of a set of defective programs known as mutants, each possessing a distinct syntactic alteration. To assess the efficacy of a given test suite, these mutants are executed using the input test set, allowing an examination of whether the injected faults can be detected.

The process of mutation testing, as delineated in prior research [40], entails generating a set of mutated programs, denoted as p', by applying predefined mutation rules to an original program P. These mutations introduce minor modifications to P, thereby creating a collection of mutants for evaluation. The determination of whether a mutant p' is classified as "killed" or "survived" is contingent upon the disparity observed in the test result between p' and the original program. More specifically, a mutant is categorized as "killed" if the test case yields a different behavior compared to that of the original test. The killing of a mutant indicates that the corresponding test case has successfully identified and flagged a potential defect in the code under examination. This discrepancy in behavior suggests that the test case has effectively detected and indicated the presence of a possible defect within the code. Conversely, a mutant is regarded as "survived" if the test result remains unchanged in comparison to the original program, suggesting that the test case fails to uncover the introduced fault. When a test suite is able to "kill" many mutants, it indicates that the suite has a higher capability to detect and localize faults within the code.

Mutation testing for DNNs Researchers have introduced various approaches and tools aimed at adapting mutation testing for deep learning systems [34, 57, 78]. Notable contributions include DeepMutation [57], DeepMutation++ [34], MuNN [78], and DeepCrime [37]. DeepMutation [57] is designed to evaluate the quality of test data for deep learning (DL) systems using mutation testing. This innovative approach encompasses the creation of mutation operators at both the source and model levels, strategically introducing faults into various components such as training data, programs, and DL models. The evaluation of test data effectiveness is subsequently conducted by analyzing the detection of these introduced faults. Building upon this foundation, DeepMutation++[34] represents an advanced iteration, introducing innovative mutation operators tailored for feed-forward neural networks (FNNs) and Recurrent Neural Networks (RNNs). Notably, it possesses the capability to dynamically mutate the run-time states of an RNN. Shen *et al.* proposed MuNN [78], an intricate mutation analysis method designed explicitly for neural networks. The method establishes five mutation operators, each rooted in the distinctive characteristics of neural networks. In a remarkable stride towards practical application, Humbatova *et al.* introduced DeepCrime [37], a mutation testing tool that implements DL mutation operators based on real-world DL faults. Furthermore, Jahangirova *et al.* [39] conducted a comprehensive empirical study of DL mutation operators found in the existing literature. Their study, which includes 20 DL mutation operators such as activation function removal and layer addition, suggests that while most operators are useful, their configuration needs careful consideration to avoid rendering them ineffective.

2.3 Test Input Prioritization for DNNs

Test prioritization [83] is a critical process in software testing that seeks to establish an optimal sequence for unlabelled tests. Its core objective is to identify and prioritize potentially misclassified tests, enabling their early labelling and consequently leading to a reduction in the overall labelling cost. The majority of approaches for prioritizing tests in Deep Neural Networks (DNNs) [21, 28, 89] can be categorized into two main groups: coverage-based and confidence-based [89]. Coverage-based approaches, exemplified by CTM [94], involve the direct extension of conventional software system testing methods to the domain of DNN testing. In contrast, confidence-based approaches prioritize test inputs based on the model's level of confidence. Specifically, these methods aim to identify inputs that are likely to be misclassified by the DNN model, as indicated by the model assigning similar probabilities to each class. DeepGini [28] stands as a classic confidence-based test prioritization method that has been empirically shown to outperform existing coverage-based techniques in terms of both effectiveness and efficiency. Other confidence-based test prioritization methods, such as Vanilla Softmax, Prediction-Confidence Score (PCS), and Entropy, have also been evaluated in recent research [90]. These metrics have demonstrated efficacy in identifying potentially misclassified test inputs and can assist in guiding test prioritization efforts.

While confidence-based methods can be applied to 3D point cloud data, they have certain limitations. 3D point cloud data is typically characterized by its large-scale and highly detailed nature, typically consisting of millions or even billions of points. However, confidence-based methods, when prioritizing tests, primarily focus on the uncertainty associated with the model's classification of the test inputs, neglecting the intrinsic raw feature information contained within the point cloud data. Furthermore, point cloud data is prone to noise, which can adversely impact the reliability of confidence scores assigned by these approaches. In the presence of noise, confidence-based methods can exhibit high confidence in incorrect labels. Consequently, in such cases, tests that will be misclassified by the model are mistakenly assigned inappropriate confidence scores, resulting in them not being prioritized higher. These factors collectively contribute to the diminished performance of confidence-based methods in the context of point cloud data.

In addition to the aforementioned test prioritization methods, PRIMA [89], proposed by Wang *et al.*, employs mutation analysis to prioritize test inputs that can uncover faults. However, point cloud data represents unstructured sets of points in three-dimensional space, which makes the mutation rules of PRIMA not adapted.

3 APPROACH

3.1 Overview

In this paper, we introduce PCPrior, a novel approach tailored for test prioritization in the domain of 3D point cloud data. The overview of PCPrior is depicted in Figure 2, which provides a visual representation of its key components. Specifically, when given a test set T targeted at a DNN model M, we outline the fundamental workflow of PCPrior as follows. A more comprehensive exposition of this workflow is presented in subsequent sections.

- **Feature Generation:** In the initial stage, PCPrior generates four distinct types of features for each test $t \in T$, which are purposefully designed to capture the characteristics of 3D point cloud data. These four types of features encompass Spatial Features, Mutation Features, Prediction Features, and Uncertainty Features. In Figure 2, the matrices represent features generated from the test inputs. Here, n denotes the number of test inputs in the test set T. Since there are n tests in T, each matrix has n rows. Each row in the matrix represents a feature vector generated for a specific test. From top to bottom, the first matrix illustrates the input mutation

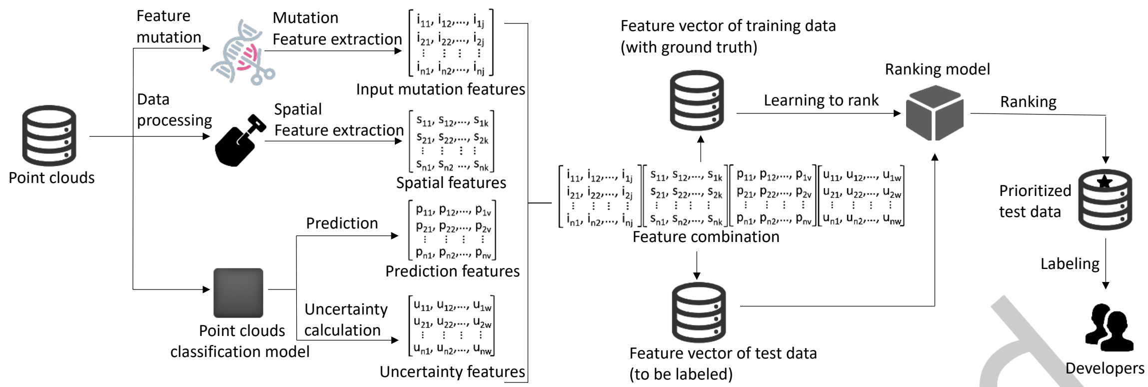


Fig. 2. Overview of PCPrior

features for all tests in the test set, with dimensions $n \times j$, where j denotes the number of mutation features for each test. The second matrix represents spatial features for all tests in the test set, having dimensions $n \times k$, where k signifies the number of spatial features for each test. The third matrix showcases prediction features, having dimensions $n \times v$, where v represents the number of prediction features for each test. The fourth matrix displays uncertainty features, with dimensions $n \times w$, where w denotes the number of uncertainty features for each test. For example, in the matrix of spatial features, $\{s_{21}, s_{22}, \ldots, s_{2k}\}$ represent all spatial features generated for the second test input in T. In Sections 3.2 to 3.5, we provide a detailed explanation of the meaning, generation methods, and motivations behind each feature type.

- **Feature Concatenation:** For each test $t \in T$, PCPrior has generated four types of features in the previous step. In this step, PCPrior concatenates these four types of features, resulting in the generation of the final feature vector specifically associated with the test t. In particular, the process is depicted in Figure 2 where four matrices are concatenated, forming a large matrix with dimensions of $n \times (j + k + v + w)$.
- **Learning to Rank:** PCPrior takes the final feature vector of each test $t \in T$ and inputs it into a pre-trained ranking model, specifically LightGBM [41]. The ranking model automatically learns the probability of misclassification for each test based on its feature vector. PCPrior leverages these probabilities to sort the tests, placing those with a higher probability of being misclassified by the model at the forefront.

3.2 Spatial Feature Generation

Based on the test set T, we generated six types of spatial features from each point cloud test input, including variance [48], mean [5], median [5], scale [74], skewness [47], and kurtosis [47]. We provide detailed explanations of each feature below. PCPrior leverages the spatial features of tests to identify their spatial proximity. As illustrated in Figure 2, prior to the generation of spatial features, a data processing step is executed. This step encompasses the reading and transformation of the point cloud dataset. Upon accessing the point cloud data, the coordinates of each point within the point cloud (commonly represented as x, y, z coordinates), along with any supplementary attributes (such as color and intensity), are transformed into a numpy array format.

The rationale behind generating these features stems from the observation made by Ma *et al.* [57] that misclassified inputs typically locate near the decision boundary of a DNN model. In light of this observation, our approach entails the generation of a diverse set of spatial features from each test input, effectively capturing its unique characteristics. As a result, each test instance is transformed into a spatial feature vector, indirectly reflecting the test's proximity to the decision boundary. Tests that exhibit closer proximity to the decision boundary are considered more susceptible to being predicted incorrectly. Motivated by this insight, PCPrior utilizes the spatial features of test inputs to assess their probability of being misclassified. For a given point cloud

P, Formula 1 illustrates the process of generating its spatial features (SF).

$$V_{SF} = Concat(\sigma^2(P), \mu(P), Median(P), Scale(P), Skewness(P), Kurtosis(P)) \quad (1)$$

In Formula 1, all feature computations rely on the coordinates of points in the point cloud P along the three coordinate axes (x, y, z). $V_{spatial}$ represents the resulting spatial feature vector for the point cloud P. Below, we use variance features as a specific example to clarify the calculation process for each type of spatial feature. Assuming P consists of five points with coordinates along the x, y, and z axes denoted as $[x_1, x_2, x_3, x_4, x_5]$, $[y_1, y_2, y_3, y_4, y_5]$, and $[z_1, z_2, z_3, z_4, z_5]$ respectively, the variance for the x-axis is calculated as $\text{var}([x_1, x_2, x_3, x_4, x_5]) = 0.5$, for the y-axis as $\text{var}([y_1, y_2, y_3, y_4, y_5]) = 0.3$, and for the z-axis as $\text{var}([z_1, z_2, z_3, z_4, z_5]) = 0.8$. Consequently, the final variance feature vector of P is $[0.5, 0.3, 0.8]$. Similar computations are performed for other types of spatial features. Specifically, in Formula 1, $\sigma^2(P)$ represents the variance features of all points in the point cloud P along each coordinate axis. $\mu(P)$ represents the mean features, $\text{Median}(P)$ denotes the median features, $\text{Scale}(P)$ represents scale features, $\text{Skewness}(P)$ indicates the skewness features, and $\text{Kurtosis}(P)$ corresponds to the kurtosis features.

- **Variance Features** [48] Variance features serve as statistical indicators for measuring the variability or dispersion of points within a dataset. They quantify the variances of point cloud data along each coordinate axis, thereby providing crucial information about the spatial distribution of points. Given a point cloud P consisting of N points, where the x, y, z coordinates of each point are respectively $X = [x_1, x_2, \ldots, x_n]$, $Y = [y_1, y_2, \ldots, y_n]$, and $Z = [z_1, z_2, \ldots, z_n]$, Formula 2 illustrates the computation process for the variance feature vector of the point cloud P.

$$\sigma^2(P) = [var(X), var(Y), var(Z)] \quad (2)$$

  where $var(X)$ represents the variance of the X-coordinates of all points in the point cloud P. Namely, it is the variance of $X = [x_1, x_2, \ldots, x_n]$. $var(Y)$ represents the variance of $Y = [y_1, y_2, \ldots, y_n]$, and $var(Z)$ represents the variance of $Z = [z_1, z_2, \ldots, z_n]$. Formula 3 precisely illustrates the computation process for the variance of X-coordinates. The procedures for computing the variance of Y and Z coordinates follow a similar approach.

$$var(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \quad (3)$$

  where $var(X)$ represents the variance of the X-coordinates of all points in P. Specifically, it is the variance of $X = [x_1, x_2, \ldots, x_n]$. μ represents the mean of the X-coordinates of all points in P. N denotes the total number of points in P. x_i represents the X-coordinates of the i-th point in P.

  Specifically, the utilization of variance features in point cloud analysis offers notable benefits: **1) Quantifying dispersion** Variance features enable a quantitative assessment of the dispersion of point cloud data along different coordinate axes. Larger variance values indicate a more scattered distribution of points along the corresponding axis, while smaller variance values suggest a more concentrated distribution. These insights are essential for comprehending the spatial characteristics and shape of the point cloud. **2) Extracting shape information** Variance features facilitate the extraction of rough shape information from the point cloud. By comparing the variances along different coordinate axes, conclusions can be drawn regarding the extension or distribution of the point cloud in various directions. For instance, if the variance along a particular axis significantly surpasses that of the other axes, it implies a greater extension of the point cloud's shape in that specific direction.
- **Mean Features** [5] In the context of 3D point cloud data, mean features refer to the feature values obtained by averaging the attributes (such as coordinates, normals, etc.) of each point in the point cloud. They represent the average attributes of the entire point cloud and provide information about the overall shape or other properties. Given a point cloud P consisting of N points, with the x, y, and z coordinates of each point represented as $X = [x_1, x_2, \ldots, x_n]$, $Y = [y_1, y_2, \ldots, y_n]$, and $Z = [z_1, z_2, \ldots, z_n]$, Formula 4 demonstrates the calculation

process for the mean feature vector of the point cloud P.

$$\mu(P) = [mean(X), mean(Y), mean(Z)] \tag{4}$$

where $mean(X)$ denotes the mean of $X = [x_1, x_2, \ldots, x_n]$, $mean(Y)$ represents the mean of $Y = [y_1, y_2, \ldots, y_n]$, and $mean(Z)$ signifies the mean of $Z = [z_1, z_2, \ldots, z_n]$. Formula 5 details the computation process for the mean of X-coordinates. The procedures for calculating the mean of Y and Z coordinates follow a similar approach.

$$mean(X) = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{5}$$

where $mean(X)$ represents the mean of the X-coordinates of all points in P. Specifically, it is the mean of $X = [x_1, x_2, \ldots, x_n]$. N denotes the total number of points in P. x_i represents the X-coordinates of the i-th point in P.

We utilize mean features in test prioritization for the following reasons: **1) Comprehensive nature** Mean features consolidate the information of the entire point cloud into a single feature vector, offering comprehensive insights about the overall characteristics. Such comprehensive features facilitate a rapid understanding of the global properties of the point cloud. **2) Dimensionality reduction** Point cloud data typically comprise a large number of points, each potentially possessing multiple attributes. By employing mean features, the point cloud data can be reduced from a high-dimensional space to a lower-dimensional feature vector, thereby reducing computational complexity and memory consumption.

- **Median Features** [5] In the context of 3D point cloud data, median features pertain to the feature values obtained by calculating the median of the coordinate attributes (X, Y, Z) within the point cloud. They serve as indicators of the central tendency of attribute values within the point cloud.

  Given a point cloud P comprising N points, where the x, y, and z coordinates of each point are denoted as $X = [x_1, x_2, \ldots, x_n]$, $Y = [y_1, y_2, \ldots, y_n]$, and $Z = [z_1, z_2, \ldots, z_n]$, Formula 6 elucidates the computation process for the median feature vector of the point cloud P.

$$Median(P) = [median(X), median(Y), median(Z)] \tag{6}$$

  where $median(X)$ refers to the median of the X-coordinates of all points in the point cloud P (i.e., $X = [x_1, x_2, \ldots, x_n]$), $median(Y)$ represents the median of $Y = [y_1, y_2, \ldots, y_n]$, and $median(Z)$ signifies the median of $Z = [z_1, z_2, \ldots, z_n]$. Formula 7 precisely outlines the computation process for the median of X-coordinates. The procedures for calculating the median of Y and Z coordinates follow a similar approach.

$$median(X) = \begin{cases} X\left(\frac{N+1}{2}\right) & \text{if } N \text{ is odd} \\ \frac{X\left(\frac{N}{2}\right)+X\left(\frac{N}{2}+1\right)}{2} & \text{if } N \text{ is even} \end{cases} \tag{7}$$

  where median(X) represents the median of the X-coordinates of all points in P. Specifically, it is the median of $X = [x_1, x_2, \ldots, x_n]$. N denotes the total number of points in P. $X\left(\frac{N+1}{2}\right)$ denotes the value in X located at the middle position. $X\left(\frac{N}{2}\right)$ and $X\left(\frac{N}{2}+1\right)$ represent the two values in X located at the middle positions when N is even.

  The utilization of median features is motivated by the following factors: 1) Median features exhibit reduced sensitivity to outliers compared to mean features, rendering them more reliable and capable of providing more accurate representations in the presence of extreme values. 2) Median features demonstrate heightened stability by being less influenced by variations in attribute value distributions, thereby facilitating a more consistent representation of the point cloud data.

- **Scale Features** [74] In the context of 3D point cloud data, scale features refer to the differences between the minimum and maximum values of each point in the three dimensions (X, Y, and Z) of the point cloud. Range features can provide information about the scale of the point cloud data, specifically the spatial extent of the point cloud in each dimension.

Given a point cloud P consisting of N points, where the x, y, and z coordinates of each point are represented as $X = [x_1, x_2, \ldots, x_n]$, $Y = [y_1, y_2, \ldots, y_n]$, and $Z = [z_1, z_2, \ldots, z_n]$, Formula 8 illustrates the computation process for the scale feature vector of the point cloud P.

$$Scale(P) = [scale(X), scale(Y), scale(Z)] \tag{8}$$

where $scale(X)$ denotes the scale of the X-coordinates of all points in the point cloud P (i.e., $X = [x_1, x_2, \ldots, x_n]$), $scale(Y)$ represents the scale of the Y-coordinates (i.e., $Y = [y_1, y_2, \ldots, y_n]$), and $scale(Z)$ signifies the scale of the Z-coordinates (i.e., $Z = [z_1, z_2, \ldots, z_n]$). Formula 9 precisely delineates the computation process for the scale of X-coordinates. The procedures for calculating the scale of Y and Z coordinates follow a similar approach.

$$scale(X) = max(X) - min(X) \tag{9}$$

where $scale(X)$ represents the scale of the X-coordinates of all points in P. $max(X)$ represents the maximum value in $X = [x_1, x_2, \ldots, x_n]$. $min(X)$ represents the minimum value in it.

The utilization of scale features lies in that they serve as descriptive features of the point cloud data, providing an overall characterization of the spatial attributes of the point cloud.

- **Skewness Features** [47] Within the context of 3D point cloud data, the skewness feature is a statistical measure employed to quantify the degree of skewness in the distribution of data. It assesses the extent to which the point cloud data distribution deviates from symmetry. In a point cloud P consisting of N points, with the x, y, and z coordinates of each point denoted as $X = [x_1, x_2, \ldots, x_n]$, $Y = [y_1, y_2, \ldots, y_n]$, and $Z = [z_1, z_2, \ldots, z_n]$, Formula 10 illustrates the computation process for the skewness feature vector of the point cloud P.

$$Skewness(P) = [skewness(X), skewness(Y), skewness(Z)] \tag{10}$$

where $skewness(X)$ denotes the skewness of $X = [x_1, x_2, \ldots, x_n]$, $skewness(Y)$ represents the skewness of $Y = [y_1, y_2, \ldots, y_n]$, and $skewness(Z)$ signifies the skewness of $Z = [z_1, z_2, \ldots, z_n]$. Formula 11 outlines the computation process for the skewness of the X-coordinates. The procedures for calculating the skewness of the Y and Z coordinates follow a similar approach.

$$skewness(X) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{x_i - \mu}{\sigma} \right)^3 \tag{11}$$

where μ represents the mean of $X = [x_1, x_2, \ldots, x_n]$, and σ denotes the standard deviation of X. N denotes the total number of points in P, and x_i represents the X-coordinates of the i-th point in P.

The utilization of the skewness feature is motivated by its ability to provide crucial insights into the distribution characteristics of point cloud data. Analyzing the skewness feature facilitates understanding the skewness patterns exhibited by the point cloud data along different dimensions, i.e., whether the data values are skewed towards the left or right. This enables the identification of inherent asymmetry or skewness phenomena present within the data.

- **Kurtosis Features** [47] In the domain of 3D point cloud data analysis, the kurtosis feature serves as a statistical measure for describing the peakedness and shape of the data distribution. It quantifies the sharpness and peakedness of the point cloud data distribution. Given a point cloud P consisting of N points, where the x, y, and z coordinates of each point are denoted as $X = [x_1, x_2, \ldots, x_n]$, $Y = [y_1, y_2, \ldots, y_n]$, and $Z = [z_1, z_2, \ldots, z_n]$, Formula 12 illustrates the computation process for the kurtosis feature vector of the point cloud P.

$$Kurtosis(P) = [kurtosis(X), kurtosis(Y), kurtosis(Z)] \tag{12}$$

where $kurtosis(X)$ represents the kurtosis of $X = [x_1, x_2, \ldots, x_n]$, $kurtosis(Y)$ denotes the kurtosis of $Y = [y_1, y_2, \ldots, y_n]$, and $kurtosis(Z)$ signifies the kurtosis of $Z = [z_1, z_2, \ldots, z_n]$. Formula 13 outlines the computation process for the kurtosis of the X-coordinates. The procedures for calculating the kurtosis of the Y and Z coordinates follow a similar approach.

$$kurtosis(X) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{x_i - \mu}{\sigma} \right)^4 - 3 \tag{13}$$

where μ represents the mean of $X = [x_1, x_2, \ldots, x_n]$, and σ denotes the standard deviation of X. N denotes the total number of points in P, and x_i represents the X-coordinates of the i-th point in P.
The utilization of the kurtosis feature stems from its ability to provide crucial insights into the distribution characteristics of point cloud data. By analyzing the kurtosis feature, it becomes possible to ascertain the peakedness of data values across different dimensions, thereby discerning the steepness of their distribution.

3.3 Mutation Feature Generation

Given a point cloud test set denoted as T and a DNN model denoted as M, we employ the following approach to mutate T and generate mutation features. It is important to note that each test sample is a point cloud composed of thousands of points. Figure 1 visually presents one example of point cloud.

- **Mutation generation** Initially, for each test sample (a point cloud) in the test set T, a group of points are randomly selected, and their coordinates are randomly perturbed to generate a mutated point cloud (also called a mutant). This process is repeated N times, each execution being independent and random, resulting in N mutants generated for each test $t \in T$.
- **Mutation feature generation** For $t \in T$, the previous step yields a set of mutants for it, denoted as $\{t'_1, t'_2, \ldots, t'_N\}$. PCPrior compares the predictions made by model M for the test t and each of its mutants t'_i, thereby constructing a mutation feature vector specific to test t. Specifically, if model M produces different predictions for test t and the mutant t'_i, PCPrior sets the i_{th} element of t's mutation feature vector to 1; otherwise, it is set to 0. PCPrior constructs a mutation feature vector for each $t \in T$. Given a test input t (a point cloud), Formula 14 describes the above mutation feature (MF) generation process in PCPrior.

$$V_{MF}[k] = \begin{cases} 1 & \text{if } M(t_k) \neq M(t) \\ 0 & \text{if } M(t_k) = M(t) \end{cases} \tag{14}$$

where V_{MF} represents the mutation feature vector generated for the test input t. $V_{MF}[k]$ denotes the k-th value of this feature vector. $M(t_k)$ represents the prediction of the 3D shape classification model M for mutant t_k, and $M(t)$ represents the prediction for the original test input t.

The principle behind leveraging the mutation features of test inputs for test prioritization is that: *A test input t is considered more likely to be misclassified if the evaluated model's predictions for many mutants of t differ from the prediction for t*. This principle draws inspiration from mutation testing techniques employed in traditional software engineering [54, 79]. Besides, our mutation feature generation approach offers several advantages:

❶ **Capturing Model Sensitivity.** The mutation feature generation approach allows to capture the sensitivity of model M to perturbations in the test input. By comparing the predictions of model M for the original test t and its corresponding mutants t'_i, we can identify test instances where even small changes in the input result in different model predictions. Such instances are considered more likely to be misclassified by the DNN model.

❷ **Fine-Grained Analysis.** By constructing a mutation feature vector specific to each test $t \in T$, we obtain a fine-grained analysis of the model's behavior for individual test cases. The mutation feature vector captures the differences between the original test and each of its mutants.

❸ **Interpretability.** The mutation feature vectors provide an interpretable representation of the model's behavior. Each element of the vector indicates whether the evaluated model's result for a specific mutant differs from its result for the original test.

3.4 Prediction Feature Generation

The Prediction Feature (PF) captures the probability information of a given test sample belonging to each class. To obtain the Prediction Feature (PF) for a test $t \in T$, initially, we input this test into the target prediction model M. This model is the 3D shape classification model that we evaluated. The model outputs a vector for the test t, denoted as $\{p_1, p_2, \ldots, p_n\}$, where this vector represents the probabilities of test t belonging to each category. Here, p_i denotes the model's prediction probability for test t belonging to category i. For instance, a feature vector [0.1, 0.1, 0.8] signifies that, according to the predictions made by model M, the test input t has a 10% probability of belonging to the first class, a 10% probability of belonging to the second class, and an 80% probability of belonging to the third class. The utilization of Prediction Features has been observed in several prior studies focusing on DNN test optimization, such as Li *et al.*[51] and Feng *et al.*[28].

Given 3D shape classification model M and a test input t, the prediction feature vector of t is obtained based on Formula 15.

$$V_{PF}(t) = M(t) = \langle p_{t,1}, p_{t,2}, \cdots, p_{t,C} \rangle \tag{15}$$

where $M(t)$ denotes the prediction probability vector of model M for the test t. $p_{t,i}$ represents the probability predicted by model M that the test input t belongs to the i-th category. C signifies the total number of predicted categories by the model M.

3.5 Uncertainty Feature Generation

The Uncertainty Features (UF) capture the model's confidence associated with its classification results for each test input $t \in T$. To obtain the UF, we employ six widely used confidence-based metrics [28, 85, 90], namely DeepGini, Vanilla SM, PCS, Entropy, Margin, and Least Confidence. These metrics are selected due to their extensive adoption in quantifying uncertainty in DNN classification tasks and their demonstrated effectiveness [33, 90]. The process of constructing the uncertainty feature vector for each test input $t \in T$ is as follows:

- **Confidence score calculation** We calculate the confidence scores for each test input t using the aforementioned six confidence-based metrics.
- **Feature generation** The uncertainty feature vector is generated by concatenating the obtained confidence scores from the six metrics. Consequently, for each test $t \in T$, a feature vector $[S_1, S_2, S_3, S_4, S_5, S_6]$ is built, where each element S_i represents the confidence score calculated by the i_{th} confidence-based metric for the test input t.

For a given test input t, Formula 16 outlines the process of generating its uncertainty features (UF). In Formula 16, DeepGini(t) denotes the uncertainty score calculated by DeepGini [28] specifically for the test t, whereas the remaining terms represent uncertainty scores computed by other metrics for measuring uncertainty.

$$V_{UF}(t) = Concat(DeepGini(t), Margin(t), Entropy(t), LC(t), Vanilla(t), PCS(t)) \tag{16}$$

3.6 Feature Concatenation

For each test input $t \in T$, we integrate four distinct types of features, namely Spatial Features (SF), Mutation Features (MF), Prediction Features (PF), and Uncertainty Features (UF), to construct a final representative feature vector. This feature vector encompasses the relevant information extracted from all feature types associated with the given test input. Subsequently, the constructed feature vector is fed into the ranking models, which are designed to evaluate the likelihood of misclassification for the test input based on its final feature vector. In the subsequent section, we provide a detailed exposition of the methodology employed in the ranking model.

3.7 Learning-to-rank

In this step, we employ the LightGBM ranking model [41] to leverage the feature vector of a given test instance $t \in T$ in order to predict its misclassification score. LightGBM is a well-regarded machine learning algorithm based on the Gradient Boosting Decision Tree (GBDT) methodology, renowned for its effectiveness and accuracy. However, due to the binary nature of LightGBM's output, which is not aligned with our objective of estimating the probability of misclassification for a test input, we introduce certain modifications to the original LightGBM algorithm. More specifically, rather than obtaining a binary classification output from the ranking models, which indicates whether the test will be predicted incorrectly, we extract the intermediary output. This intermediate result conveys valuable information regarding the probability of misclassification for each test input.

Upon completion of the training phase (described in Section 3.8) of LightGBM, when a feature vector of a test instance is provided as input to the ranking model, we extract an intermediate value predicted by LightGBM. In the following, we provide a detailed explanation of how the intermediate value is obtained from LightGBM. Initially, the original LightGBM was a binary classification model. For a given test, it can categorize the test into two classes based on its final feature vector (obtained from the above steps), where an output of 0 indicates that the test will be correctly predicted by the model, and an output of 1 indicates that the test will be incorrectly predicted. Its internal logic operates as follows: For a test t_i, LightGBM first generates an **intermediate value**, which signifies the probability of the test being incorrectly predicted by the model. If this intermediate value exceeds 0.5, LightGBM will classify it as 1, indicating that the test is likely to be misclassified by the model. Conversely, if the value is below 0.5, it will be classified as 0, suggesting that the test is likely to be correctly predicted. In PCPrior, rather than letting LightGBM carry out classification, we directly extract this intermediate value for the purpose of test prioritization. Tests with higher intermediate values are considered more likely to be incorrectly predicted and are, therefore, assigned a higher priority.

Specifically, when PCPrior is used for test prioritization, the detailed process of learning-to-rank is as follows: For each test $t_i \in T$, based on its final feature vector, the LightGBM model generates an intermediate value for it, which we denote as F_i, representing the probability of test t_i being predicted incorrectly by the model M. It ranges from 0 to 1. If F_i for a test is closer to 1, it indicates that the test is more likely to be predicted incorrectly by the model. PCPrior ranks all the tests in the test set based on their F_i value. Tests with higher F_i will be prioritized higher.

In the following, we explain the reasons for choosing the LightGBM model as the default model for PCPrior:

- **Improved effectiveness**. In RQ2 (cf. Section 5.2), we evaluated the effectiveness of different ranking models on test prioritization. We found that LightGBM and XGBoost performed best across all subjects. However, in most cases of our experiments, LightGBM outperformed XGBoost.
- **Faster training speed**. Moreover, prior work [41] has indicated that LightGBM trains faster than XGBoost. Therefore, compared to XGBoost, LightGBM is more efficient.

3.8 Usage of PCPrior

Through the utilization of ranking models, the PCPrior framework is able to predict a misclassification score for each test input within a given test set. These predicted scores are then employed for test prioritization, prioritizing test inputs with higher scores. Specifically, the ranking models undergo pre-training prior to the execution of PCPrior. The training process is presented as follows:

❶ **Training Set Construction:** Given a DNN model M with a point cloud dataset D, the dataset D is initially split into two partitions: the training set R and the test set T, following a 7:3 ratio [62]. The test set remains untouched to evaluate the performance of PCPrior. Based on the training set R, our objective is to build a training set R' for training the ranking models. Firstly, we generate four types of features for each training input $r_i \in R$, using the procedures described in Sections 3.2 to 3.5. Then, we obtain the final feature vector V_i for each training input r_i, following the guidelines in Section 3.6. This final feature vector is utilized to construct the training set R', which serves as the training data for the ranking models. Secondly, we input each training input $r_i \in R$ into the original model and obtain its classification results, denoted as L_i. By comparing L_i with the ground truth of r_i, we determine whether r_i is misclassified by the model M. If r_i is misclassified, it is labeled as 1; otherwise, it is labeled as 0. This process enables the label construction of the ranking model training set R'.

❷ **Ranking Model Training:** Using the training set R', we proceed to train the ranking models. Upon completion of the training process, the ranking model is capable of producing a misclassification score for a given input, based on the feature vector generated by PCPrior.

4 STUDY DESIGN

In this section, we provide a comprehensive exposition of the details pertaining to our study design. Specifically, Section 4.1 elucidates the research questions that served as the guiding framework for our investigation. Within Sections 4.2 and 4.3, we meticulously present the point cloud subjects and measurement metrics that were employed to assess the effectiveness of PCPrior. Furthermore, Section 4.4 showcases the five DNN test prioritization methods that were employed as comparative approaches against PCPrior. In Section 4.5, we elucidate the design and characteristics of PCPrior variants. Additionally, Section 4.6 exhibits the implementation and configuration setup that were utilized in our study.

4.1 Research Questions

Our experimental evaluation answers the research questions below.

- **RQ1: How does PCPrior perform in prioritizing test inputs for 3D point clouds?**
  In contrast to existing test prioritization methodologies, our proposed approach, PCPrior, leverages the unique characteristics of point clouds for test prioritization. In this research question, we evaluate the effectiveness of PCPrior by comparing it with existing test prioritization approaches that have been demonstrated effective in prior studies [28, 90] and random selection (baseline).
- **RQ2: How do different ranking models affect the effectiveness of PCPrior?**
  In the original implementation of PCPrior, the LightGBM ranking algorithm [41] was employed to leverage the generated features of test inputs for test prioritization. In this research question, we explore the utilization of alternative ranking algorithms, namely Logistic Regression [81], XGBoost [17], and Random Forest [9], with the objective of examining the influence of ranking models on the effectiveness of PCPrior. To this end, we design a set of variants for PCPrior, each incorporating one of the aforementioned ranking models, while maintaining consistency with the remaining workflow.
- **RQ3: How does the selection of main parameters of PCPrior affect its effectiveness?**
  We conducted an in-depth investigation of the main parameters in PCPrior, with the aim of evaluating whether PCPrior can consistently outperform the compared test prioritization approaches when these parameters undergo modifications.
- **RQ4: How does PCPrior and its variants perform on noisy 3D point clouds?**
  In addition to assessing PCPrior and its variants on natural datasets, we undertake an evaluation that encompasses noisy 3D point clouds, thereby facilitating an in-depth examination of their effectiveness.
- **RQ5: To what extent does each type of features contribute to the effectiveness of PCPrior?**

In PCPrior, we generate four different types of features from each test input for test prioritization, namely Spatial Features, Mutation Features, Prediction Features and Uncertainty Features, as elaborated in Section 3. In this research question, we compare the contributions of different types of features on the effectiveness of PCPrior.

- **RQ6: Can PCPrior and uncertainty-based methods be employed to guide the retraining process for enhancing a 3D shape classification model?**
  Faced with a substantial volume of unlabeled inputs and a constrained time budget, manually labeling all inputs for retraining a 3D shape classification model becomes impractical. Active learning is acknowledged as a practical solution for reducing data labeling costs [71]. This approach focuses on selecting an informative subset of samples to retrain the model, aiming to improve model performance with minimal labeling costs. In this research question, we investigate the effectiveness of PCPrior and uncertainty-based metrics in selecting informative retraining inputs to improve the performance of 3D shape classification models.

4.2 Models and Datasets

The effectiveness of PCPrior and the compared test prioritization approaches [28, 90] was evaluated using a set of 165 subjects. Essential details regarding these subjects are presented in Table 1, which highlights the matching relationship between the point cloud dataset and the DNN models. In particular, the "#Size" column indicates the size of the dataset, while the "Type" column denotes the type of the dataset, with "Original" representing natural data and "Noisy" indicating noisy data.

Among the 165 subjects, 15 subjects (3 point cloud datasets $\times$ 5 models) were generated using natural datasets, while the remaining 150 subjects were generated using noisy datasets. To generate a noisy dataset from the original test set T, each test instance $t \in T$ undergoes a modification. Specifically, within each test instance t (a point cloud), approximately 30% of the points undergo a random offset, while the remaining 70% of the points remain unchanged. The 30% ratio is derived from the reasonable range of noise injection proportions provided in the existing work [2]. The 150 subjects derived from noisy data were obtained as follows: For each original dataset, we generated 10 noisy datasets, resulting in a total of 30 noisy datasets. Each noisy dataset was paired with five different models, resulting in a total of 150 subjects (30 datasets $\times$ 5 models).

In the following part, we present the description of the 3D point cloud datasets and DNN models utilized in our study.

4.2.1 Datasets.

In our research, we employed three prominent point cloud datasets, namely ModelNet40 [93], ShapeNet [10], and S3DIS [4]. These datasets are widely adopted within the academic community and have consistently served as benchmarks for several state-of-the-art point cloud studies [30, 35, 52].

- **ModelNet40** [93]: ModelNet40 consists of 12,311 point clouds in 40 categories (e.g., airplane, car, plant, lamp). It encompasses synthetic object point clouds and stands as a paramount benchmark for point cloud analysis. Renowned for its diverse range of categories, meticulous geometric shapes, and methodical dataset construction, ModelNet40 has garnered significant popularity in the research community [30].
- **ShapeNet** [10]: ShapeNet dataset is a widely recognized and extensively used benchmark in the field of 3D shape classification. The ShapeNet dataset utilized in our study consists of 50 categories and a total of 53,107 samples. These categories include chairs, tables, cars, airplanes, animals, etc.
- **Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS)** [4]: The S3DIS dataset is widely recognized for its comprehensive representation of diverse indoor environments, encompassing various real-world scenes encountered in indoor settings. The S3DIS dataset utilized in our study consists of 9,813 samples, classified into 13 categories (e.g., office, meeting room, and open space).

Table 1. 3D Point cloud datasets and models

ID	Dataset	# Size	Model	Type
1	ModelNet	12311	DGCNN	Original, Noisy
2	ModelNet	12311	PointConv	Original, Noisy
3	ModelNet	12311	MSG	Original, Noisy
4	ModelNet	12311	SSG	Original, Noisy
5	ModelNet	12311	PointNet	Original, Noisy
6	S3DIS	9813	DGCNN	Original, Noisy
7	S3DIS	9813	PointConv	Original, Noisy
8	S3DIS	9813	MSG	Original, Noisy
9	S3DIS	9813	SSG	Original, Noisy
10	S3DIS	9813	PointNet	Original, Noisy
11	ShapeNet	53107	DGCNN	Original, Noisy
12	ShapeNet	53107	PointConv	Original, Noisy
13	ShapeNet	53107	MSG	Original, Noisy
14	ShapeNet	53107	SSG	Original, Noisy
15	ShapeNet	53107	PointNet	Original, Noisy

4.2.2 Models.

- **PointConv** [92]: PointConv is a convolutional neural network operator specifically designed for processing 3D point clouds characterized by non-uniform sampling. By training multi-layer perceptrons using local point coordinates, PointConv approximates continuous weight and density functions within convolutional filters. In this way, deep convolutional networks can be directly constructed on 3D point clouds, enabling efficient and effective analysis and processing.
- **Dynamic Graph Convolutional Neural Network (DGCNN)** [88]: DGCNN is a deep learning architecture specifically designed for processing and analyzing 3D point cloud data. The key idea behind DGCNN is to exploit the intrinsic spatial relationships present in point clouds by modeling them as graphs. By leveraging graph convolutions and dynamically adapting the graph structure based on the input data, DGCNN can effectively learn and process point cloud representations, making it suitable for point cloud classification tasks.
- **PointNet** [72]: PointNet is a widely-adopted deep learning architecture specifically tailored for 3D point cloud data. The architecture includes a shared multi-layer perceptron (MLP) with max-pooling to extract local features from individual points and a symmetric function to aggregate the global features across all points. By employing T-Net layers, PointNet is able to learn transformation matrices that aid in aligning and transforming input point clouds, enhancing the model's robustness to input variations. PointNet has demonstrated impressive capabilities in 3D shape classification tasks, establishing it as an effective approach for point cloud analysis.
- **MSG** [73]: MSG refers to multi-scale grouping. The MSG approach involves sampling representative points and grouping nearby points within a specified radius. This allows for the extraction of local features at multiple scales, enabling hierarchical feature learning from point sets.
- **SSG** [73]: SSG, an acronym for Single-Scale Grouping, denotes a simplified variant of the multi-scale grouping architecture. The essence of SSG lies in the partitioning of a point cloud into local regions of fixed size while disregarding the consideration of multiple scales. Within each region, a representative subset of points is judiciously sampled, and proximate points falling within a predefined radius are grouped together. This approach facilitates local feature extraction while avoiding the intricate intricacies associated with handling diverse scales.

4.3 Measurements

The goal of PCPrior is to prioritize the possibly-misclassified test inputs in the context of 3D point cloud data. Thus following the existing work [28], we adopted Average Percentage of Fault-Detection (APFD) and Percentage of Fault Detected (PFD) to measure the effectiveness of PCPrior, the compared approaches, and the variants of PCPrior.

- **Average Percentage of Fault-Detection (APFD)** APFD [94] is a widely recognized metric for assessing the effectiveness of prioritization techniques. A higher APFD value indicates a quicker rate of detecting misclassifications. The calculation of APFD values is based on Formula 17.

$$APFD = 1 - \frac{\sum_{i=1}^{k} o_i}{kn} + \frac{1}{2n} \tag{17}$$

where n denotes the total number of test inputs, and the variable k represents the number of test inputs in T that will be incorrectly predicted by the model. The index o_i pertains to the position of the i_{th} misclassified test within the prioritized test set. Specifically, o_i represents an integer value indicating the position of the i_{th} misclassified test within the prioritized test set.
Based on the existing study [28], we normalize the APFD values to [0,1]. A prioritization approach is considered better when the APFD value is closer to 1. This is because: a larger APFD value corresponds to a smaller value of $\sum_{i=1}^{k} o_i$. Here, $\sum_{i=1}^{k} o_i$ represents the total index sum of misclassified tests within the prioritized list. A smaller $\sum_{i=1}^{k} o_i$ implies that the evaluated test prioritization method assigns higher priority to misclassified tests, positioning them at the front of the ranked test set. This effective detection of misclassified tests demonstrates the efficacy of the test prioritization approach. Therefore, a larger APFD value serves as an indicator of better effectiveness for test prioritization strategies.
- **Percentage of Fault Detected (PFD)** PFD refers to the proportion of detected misclassified test inputs among all misclassified tests. Higher PFD values indicate better test prioritization effectiveness. PFD is calculated based on Formula 18.

$$PFD = \frac{\#N_d}{\#N} \tag{18}$$

where $\#N_d$ is the number of misclassified test inputs that have been detected. $\#N$ denotes the total number of misclassified tests. In our study, we evaluated the PFD of PCPior and the compared test prioritization approaches against different ratios of prioritized tests. We utilize **PFD-n** to denote the first n% prioritized test inputs.

4.4 Compared Approaches

This study employed five comparative approaches, which included a baseline approach (random selection) and four DNN test prioritization techniques. The selection of these methods was driven by multiple factors: 1) These approaches can be adapted for test prioritization in the context of 3D point cloud data; 2) These approaches were proposed within the DL testing community and have been previously demonstrated as effective for DNNs; 3) These approaches provide open-source implementations.

- **Random selection** [26] Random selection is the baseline in our study. Random selection involves the randomized determination of the execution order for test inputs. This means that the sequencing of test inputs is established in a completely arbitrary manner, devoid of any predetermined patterns or logical arrangements.
- **DeepGini** [28] DeepGini utilizes the Gini coefficient, which is a statistical metric used to assess the probability of misclassification, in order to facilitate the ranking of test inputs. The Gini score is calculated according to

Formula 19, which is presented below:

$$G(t) = 1 - \sum_{i=1}^{N} (p_i(t))^2 \tag{19}$$

where $G(t)$ represents the probability of the test input t being misclassified. $p_i(t)$ denotes the probability that the test input t is predicted to belong to label i. N represents the total amount of categories that the input can be assigned to.

- **Prediction-Confidence Score (PCS)** PCS [90] assigns rankings to test inputs based on the difference between the predicted class and the second most confident class in the softmax likelihood. A smaller difference indicates that the model is less certain about the prediction for a particular test input. These uncertain tests are given higher priority and are placed at the front of the test set. The calculation of this difference is defined by Formula 20 as follows:

$$P(x) = l_k(x) - l_j(x) \tag{20}$$

where $l_k(x)$ refers to the most confident prediction probability. $l_j(x)$ refers to the second most confident prediction probability.

- **Vanilla Softmax** [90] Vanilla Softmax measures the difference between the maximum activation probability in the output softmax layer and the ideal value of 1 for each test input. This disparity reflects the degree of uncertainty associated with the model's predictions. Test inputs with larger disparities are considered more likely to be misclassified by the model. The specific computation of this disparity is illustrated by Formula 21, which provides a clear and concise representation of the underlying mathematical calculations.

$$\mathrm{V}(x) = 1 - \max_{c=1}^{C} l_c(x) \tag{21}$$

where $l_c(x)$ belongs to a valid softmax array in which all values are between 0 and 1, and their sum is 1.

- **Entropy** [90] Entropy serves as a criterion for ranking test inputs based on the entropy of their softmax likelihood. Higher entropy values indicate greater uncertainty in the model's predictions for those inputs. Consequently, test inputs with higher entropy are considered more likely to be misclassified by the model. As a result, they are given higher priority and placed at the beginning of the test set.

4.5 Variants of PCPrior

We conducted an investigation into the influence of different ranking models on the effectiveness of PCPrior. To this end, we proposed five variants of PCPrior, namely PCPrior^L, PCPrior^X, PCPrior^R, PCPrior^D, and PCPrior^T, which utilize Logistic Regression [81], XGBoost [17], Random Forest [9], DNNs [82], and TabNet [3] as the ranking model, respectively. It is essential to emphasize that apart from the variation in ranking models, the execution workflow of these derived variants remains identical to that of the original PCPrior approach.

Furthermore, we extended the modifications applied to the LightGBM ranking model of PCPrior to the ranking models employed by the variants of PCPrior. Specifically, instead of making the ranking models provide a binary classification output (i.e., indicating whether the test will be predicted incorrectly by the model), we extract the intermediate output, which can indicate the probability of misclassification for each test input. Consequently, we obtain a misclassification score for each test input, which can be effectively utilized for test prioritization. In the following sections, we provide a comprehensive explanation of the specific ranking models utilized in each variant of PCPrior.

- **PCPrior^L:** In the context of PCPrior^L, we employ the Logistic Regression algorithm [61] as the ranking model. Logistic Regression is a statistical modeling technique that employs a logistic function to establish the relationship between a categorical dependent variable and one or more independent variables.

- **PCPrior^X:** In the context of PCPrior^X, we utilize the XGBoost ranking algorithm [17] to estimate the misclassification score of a test input based on its corresponding feature vector. XGBoost is a powerful gradient-boosting technique that integrates decision trees to enhance prediction accuracy. By leveraging its ensemble learning capabilities, XGBoost effectively captures complex relationships within the data, enabling accurate estimation of the likelihood of misclassification for each test input.
- **PCPrior^R:** In the context of PCPrior^R, we employ the Random Forest algorithm [9] as the ranking model. Random Forest is an ensemble learning algorithm that constructs multiple decision trees. The predictions from individual trees are combined using averaging or voting mechanisms to produce the final prediction. Random Forest is known for its ability to handle high-dimensional data and capture intricate interactions among features. By leveraging these strengths, PCPrior^R accurately estimates the misclassification score for each test input, aiding in effective test prioritization.
- **PCPrior^D:** In the context of PCPrior^D, we utilize a DNN model as the ranking model, derived from a prior investigation [82]. This DNN model is capable of producing a misclassification score for a given test input, relying on its feature vector generated by PCPrior.
- **PCPrior^T:** In the context of PCPrior^T, we utilize TabNet [3] as the ranking model. TabNet is a DNN architecture specifically designed for tabular data. It has been demonstrated to be more effective than XGBoost and LightGBM in a previous study [3].

4.6 Implementation and Configuration

We implemented PCPrior in Python, utilizing the PyTorch 2.0.0 framework [68]. To enable comparison with other approaches, we integrated existing implementations of the compared methods [28, 90] into our experimental pipeline, specifically tailored for test prioritization of 3D point cloud data. To generate mutation features, we created 30 mutants for each test sample. Regarding the configuration of the ranking models employed in PCPrior, we utilized XGBoost 1.7.4, LightGBM 3.3.5, and scikit-learn 1.0.2 frameworks. Furthermore, we made specific parameter selections: for LightGBM, the learning rate was set to 0.1; for Logistic Regression, the parameter max_iter was set to 100; for XGBoost, the learning rate was set to 0.3; and for the random forest algorithm, the number of estimators was set to 100. Our experimental setup involved conducting experiments on NVIDIA Tesla V100 32GB GPUs. For the data analysis, we utilized a MacBook Pro laptop running Mac OS Big Sur 11.6, equipped with an Intel Core i9 CPU and 64 GB of RAM. In total, we conducted experiments on 165 subjects, consisting of 15 subjects based on natural inputs and 150 subjects based on noisy inputs.

5 RESULTS AND ANALYSIS

5.1 RQ1: Performance of PCPrior

Objectives: We investigate the effectiveness and efficiency of PCPrior, comparing it with several existing test prioritization approaches.
Experimental design: We conducted experiments to evaluate the performance of PCPrior from the following three aspects.

- **Effectiveness evaluation on natural datasets.** We employed a set of 15 subjects constructed from 3D point cloud datasets to evaluate the effectiveness of PCPrior. Table 1 presents the basic information of these subjects. In order to assess the performance of PCPrior, we carefully selected four test prioritization approaches, namely DeepGini, Vanilla SM, PCS, and entropy, alongside a baseline method (i.e., random selection), for comparative analysis. Moreover, we utilized two measurement metrics, specifically the Average Percentage of Fault-Detection (APFD) and the Percentage of Fault Detected (PFD), to evaluate the effectiveness of PCPrior and the compared approaches. A detailed explanation of the calculations for these metrics can be found in Section 4.3.

Table 2. Effectiveness comparison among PCPrior, DeepGini, VanillaSM, PCS, Entropy and random selection in terms of the APFD values on natural datasets

Approach	ModelNet					S3DIS					ShapeNet				
	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet
Random	0.503	0.503	0.489	0.482	0.509	0.514	0.493	0.505	0.509	0.504	0.502	0.495	0.496	0.501	0.511
DeepGini	0.763	0.719	0.757	0.748	0.734	0.715	0.700	0.699	0.703	0.653	0.846	0.740	0.824	0.837	0.793
VanillaSM	0.768	0.725	0.763	0.754	0.737	0.718	0.702	0.702	0.705	0.657	0.850	0.745	0.827	0.839	0.797
PCS	0.770	0.729	0.767	0.756	0.737	0.717	0.699	0.700	0.702	0.657	0.853	0.746	0.830	0.841	0.800
Entropy	0.751	0.707	0.743	0.735	0.724	0.703	0.692	0.690	0.696	0.644	0.831	0.728	0.813	0.829	0.780
PCPrior	0.807	0.781	0.809	0.796	0.793	0.833	0.827	0.815	0.820	0.817	0.905	0.852	0.897	0.904	0.891

- **Statistical analysis.** Due to the inherent randomness in the model training process, we performed statistical analysis by conducting the experiments ten times. Specifically, for each subject, which refers to a point cloud dataset paired with a DNN model, we generated ten distinct models through separate training processes. The average results are reported in our experimental findings. Moreover, for each subject, we calculated the variance of ten repeated experimental results for each test prioritization method to demonstrate the stability of PCPrior better.
  To further validate the stability and reliability of the experimental findings, we calculated p-values associated with the results. Specifically, we employed the **paired two-sample t-test** [46] to calculate the p-value, a commonly used statistical method for evaluating differences between two related data sets. The essential steps involved are: 1) selecting two related sets of data, 2) computing the difference for each corresponding pair of data points, and 3) analyzing these differences to ascertain if there is a statistically significant disparity between the two data sets. In the paired two-sample t-test approach, the significance of the results is determined by the p-value. Generally, if the p-value is less than 10^{-05}, it is considered that the difference between the two sets of data is statistically significant [58]. Additionally, we quantify the magnitude of the difference between the two sets of results through the *Effect Size*. Specifically, we use Cohen's d for measuring the effect size [42]. Wherein, $|d| < 0.2$ – "negligible," $|d| < 0.5$ – "small," $|d| < 0.8$ – "medium," otherwise – "large". To ensure that the difference between the results of PCPrior and the compared approach is "non-negligible", we require that the value of d is greater than or equal to 0.2.
- **Efficiency evaluation.** In addition to evaluating the effectiveness of PCPrior, we conducted an assessment of its efficiency and compared it with the selected test prioritization methods. Specifically, we quantified the time required for each step of PCPrior to measure its efficiency. By analyzing the execution time of PCPrior, we aim to gain insights into its computational efficiency and its potential for practical application in real-world scenarios.

Results: The experimental findings pertaining to Research Question 1 (RQ1) are presented in Table 2, Table 3, Table 4, Table 5, Table 6, Table 7 and Figure 3. Table 2 and Table 3 offer a comparative analysis, employing the APFD metric, between PCPrior and the comparative methods. Conversely, Table 6 and Figure 3 provide an assessment of effectiveness using the PFD metric. It is important to note that we highlight the approach with the highest effectiveness for each case in grey. Additionally, Table 7 offers a comparison of the efficiency between PCPrior and the evaluated test prioritization approaches.

Notably, Table 2 reveals that across all 15 subjects, PCPrior consistently outperforms all comparative methods in terms of its APFD. Specifically, the range of APFD values for PCPrior spans from 0.781 to 0.905, while the range for the comparative methods lies between 0.495 and 0.853. Moreover, Table 3 further highlights the average APFD value for PCPrior and its relative improvement compared to the comparative methods. We see that PCPrior achieves an average APFD of 0.836, whereas the average APFD of the comparative methods falls within the range of 0.501 to 0.754. The improvement observed in PCPrior, relative to the comparative methods, ranges from 10.99% to 66.94%. These findings demonstrate that PCPrior performs better than all the comparative test prioritization methods in terms of the APFD metric.

Table 3. Effectiveness improvement of PCPrior over the compared approaches in terms of the APFD values on natural datasets

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.501	66.94
DeepGini	0	0.749	11.72
VanillaSM	0	0.753	11.14
PCS	0	0.754	10.99
Entropy	0	0.738	13.38
PCPrior	15	0.836	-

Table 4. Statistical analysis on natural test inputs (in terms of p-value and effect size)

	Random	DeepGini	VanillaSM	PCS	Entropy
PCPrior (p-value)	3.444×10^{-14}	2.039×10^{-07}	4.403×10^{-07}	8.663×10^{-07}	3.071×10^{-08}
PCPrior (effect size)	7.854	2.423	2.273	2.148	2.822

Table 5. Variance in experimental results ($\times 10^{-3}$) for PCPrior and the compared approaches across ten repetitions

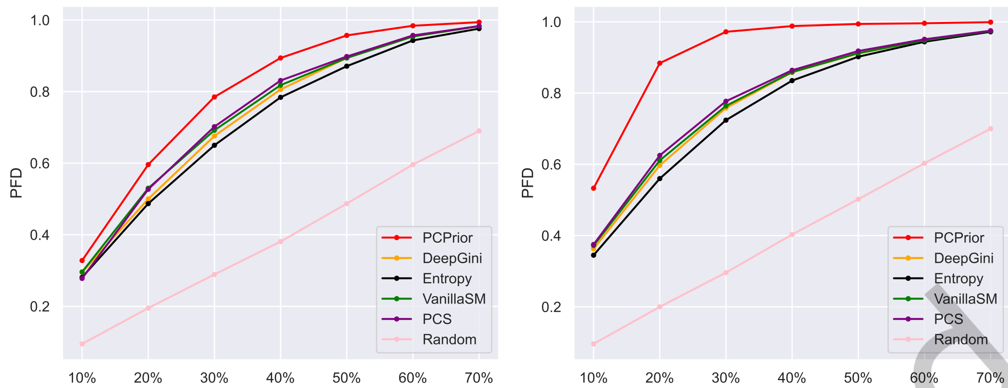
Approach	ModelNet					S3DIS					ShapeNet				
	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet
Random	0.044	0.012	0.056	0.026	0.054	0.079	0.036	0.102	0.089	0.035	0.037	0.008	0.018	0.047	0.021
DeepGini	0.026	0.200	0.050	0.021	0.031	0.071	0.405	0.026	0.045	0.038	0.027	0.064	0.015	0.009	0.035
VanillaSM	0.022	0.196	0.042	0.022	0.025	0.071	0.396	0.027	0.052	0.029	0.025	0.065	0.015	0.010	0.036
PCS	0.013	0.209	0.031	0.024	0.022	0.068	0.430	0.030	0.051	0.024	0.022	0.070	0.015	0.012	0.033
Entropy	0.030	0.199	0.061	0.021	0.039	0.075	0.404	0.019	0.036	0.053	0.035	0.054	0.020	0.008	0.033
PCPrior	0.005	0.112	0.022	0.011	0.042	0.030	0.375	0.014	0.017	0.030	0.008	0.073	0.003	0.001	0.002

Table 6. Average comparison results among PCPrior and the compared approaches on natural data in terms of PFD

Data	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
ModelNet	Random	0.100	0.206	0.300	0.398	0.498	0.602	0.699
	DeepGini	0.263	0.467	0.641	0.774	0.874	0.935	0.973
	VanillaSM	0.269	0.483	0.658	0.785	0.875	0.936	0.973
	PCS	0.261	0.488	0.664	0.794	0.881	0.938	0.974
	Entropy	0.253	0.452	0.612	0.746	0.851	0.923	0.968
	PCPrior	0.305	0.567	0.760	0.882	0.950	0.983	0.994
S3DIS	Random	0.101	0.202	0.297	0.400	0.499	0.602	0.705
	DeepGini	0.222	0.402	0.554	0.684	0.789	0.873	0.929
	VanillaSM	0.228	0.409	0.563	0.688	0.790	0.874	0.929
	PCS	0.222	0.410	0.556	0.690	0.789	0.875	0.928
	Entropy	0.212	0.391	0.535	0.663	0.775	0.867	0.926
	PCPrior	0.341	0.629	0.829	0.931	0.972	0.989	0.995
ShapeNet	Random	0.099	0.200	0.297	0.395	0.495	0.597	0.694
	DeepGini	0.386	0.632	0.789	0.878	0.928	0.959	0.979
	VanillaSM	0.399	0.647	0.793	0.879	0.928	0.959	0.979
	PCS	0.403	0.656	0.801	0.884	0.932	0.960	0.979
	Entropy	0.368	0.597	0.758	0.860	0.919	0.955	0.977
	PCPrior	0.555	0.865	0.961	0.984	0.992	0.996	0.998

The comparative analysis presented in Table 6 employs the PFD metric to exhibit the comparison between PCPrior and various DNN test prioritization methods. Notably, from prioritizing 10% to 70% of the dataset, PCPrior consistently outperforms all comparative methods in terms of PFD. To facilitate a more intuitive comparison, Figure 3 showcases two line graphs with PFD as the y-axis, illustrating the cases of ModelNet dataset with DGCNN model and ShapeNet dataset with PointNet model, respectively. All the results can be found on our Github[2].

[2]https://github.com/yinghuali/PCPrior/tree/main/result



(a) ModelNet, DGCNN

(b) ShapeNet, PointNet

Fig. 3. Test prioritization effectiveness among PCPrior and the compared approaches for ModelNet with DGCNN and ShapeNet with PointNet. X-Axis: the percentage of prioritized tests; Y-Axis: the percentage of detected miscalssified tests.

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

Time cost	Approach					
	PCPrior	Random	DeepGini	VanillaSM	PCS	Entropy
Feature generation	6 min	-	-	-	-	-
Ranking model training	32 s	-	-	-	-	-
Prediction	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s

In the figures, PCPrior is depicted by the red lines, while the baseline is represented by the pink lines. Visual analysis reveals that PCPrior consistently achieves a higher PFD value when contrasted with DeepGini, entropy, Vanilla SM, PCS, and random methods. These experimental results demonstrate that PCPrior outperforms all comparative test prioritization methods in terms of the PFD metric.

As stated in the experimental design, a statistical analysis was conducted to ensure the stability of our findings. To this end, all experiments were repeated ten times for each subject. The statistical analysis reveals a p-value lower than 10^{-05}, providing strong evidence that PCPrior consistently outperforms the compared approaches in the context of test prioritization. Table 4 presents detailed results from the statistical analysis. The analysis employs two primary metrics: p-value and effect size. As outlined in the experimental design, a p-value less than 10^{-05} indicates that the difference between two data sets is statistically significant [58]. Furthermore, an effect size ≥ 0.2 suggests that the difference is "non-negligible." In Table 4, we observed that all the p-values between PCPrior and the compared approaches consistently fall below 10^{-05}, indicating that PCPrior statistically outperforms all the compared test prioritization methods. For example, the p-value for the difference in experimental results between PCPrior and DeepGini is 2.039×10^{-07}. The p-value between PCPrior and VanillaSM is 4.403×10^{-07}. Additionally, the experimental results for both PCPrior and the compared approaches show effect sizes exceeding 0.2, confirming a non-negligible difference. Moreover, we found that all the effect sizes are even greater than 0.8. For example, the effect size of PCPrior and VanillaSM is 2.273. According to Cohen's d [42], this means that the difference in experimental results between PCPrior and the compared methods is not only statistically significant but also relatively "large" in scale.

Moreover, for each case, we calculated the variance of ten repeated experimental results with respect to each test prioritization method, as presented in Table 5. It is important to note that the unit for the table is 10^{-3}. For instance, in the second row, the first number, 0.026, represents that for the ModelNet dataset, under the DGCNN model, the variance of ten repeated experimental results for the DeepGini method is 0.026×10^{-3}. The cases highlighted in grey represent the test prioritization method with the minimum variance for each subject. We see that for 66.7% (10 out of 15) of subjects, PCPrior has the smallest variance. Furthermore, the variance range for PCPrior is 0.001×10^{-3} to 0.375×10^{-3}. In contrast, the variance range for comparative methods is 0.008×10^{-3} to 0.430×10^{-3}. The above experimental results indicate that the variance of PCPrior's results is generally lower compared to the comparative test prioritization methods, suggesting that PCPrior is relatively more stable.

Table 7 provides a comprehensive comparison of the efficiency between PCPrior and the compared test prioritization approaches. A noteworthy distinction between our proposed method and the comparative approaches pertains to the requirement of training a ranking model and generating features. As can be observed from Table 7, the overall time taken by PCPrior is approximately 6 minutes and 32 seconds. Specifically, the average training time for the PCPrior ranking model amounts to 32 seconds, while the average time for feature generation is 6 minutes. The final prediction time of the compared approaches is less than 1s. Although PCPrior is not as efficient as confidence-based test prioritization approaches, the effectiveness improvement of PCPrior relative to confidence-based methods is 10.99%~13.38%. Considering the trade-off between effectiveness and efficiency, PCPrior remains a practical option.

Answer to RQ1: *PCPrior consistently demonstrates better performance compared to all the evaluated test prioritization approaches (i.e., DeepGini, Vanilla SM, PCS, Entropy, and Random) in the field of test prioritization for 3D point cloud data, as assessed by both the APFD and PFD metrics. Specifically, the average improvement achieved in terms of APFD ranges from 10.99% to 66.94%. While PCPrior is not as efficient as confidence-based methods, considering the trade-off between effectiveness and efficiency, it remains a practical option.*

5.2 RQ2: Influence of ranking models

Objectives: We investigate the impact of various ranking models on the effectiveness of PCPrior.
Experimental design: We proposed five variants of PCPrior that incorporate different ranking models. In addition to the ranking models, the other procedures of these methods remain identical to PCPrior. The five variants are PCPrior^L, PCPrior^X, PCPrior^R, PCPrior^D, and PCPrior^T, which utilize Logistic Regression [81], XGBoost [17], Random Forest [9], DNNs [82], and TabNet [3] as the ranking model, respectively. We evaluated the impact of these ranking models on the effectiveness of PCPrior by assessing the performance of these variants on natural datasets utilizing both the APFD and PFD metrics.
Results: The experimental results for Research Question 2 (RQ2) are presented in Table 8 and Table 9. Table 8 showcases the comparison between PCPrior and its variants in terms of the APFD metric, while Table 9 presents their comparison based on the PFD metric.

In Table 8, we see that PCPrior, which employs LightGBM as the ranking model, performs the best in 66.67% (10 out of 15) of the cases. PCPrior^X, which utilizes XGBoost as the ranking model, performs the best in the remaining 33.3% (5 out of 15) cases. Furthermore, Table 9 presents a comparison of the effectiveness of PCPrior and its variants from the perspective of the PFD metric. We see that PCPrior performs the best in 61.9% (13 out of 21) cases, while PCPrior^X performs the best in 38.1% (8 out of 21) of the cases. The aforementioned experimental results illustrate that the ranking models employed by PCPrior and PCPrior^X, specifically LightGBM and XGBoost, can better utilize the generated test input features for test prioritization.

Surprisingly, despite existing studies [3] mentioning that TabNet is more effective than XGBoost and LightGBM in their evaluated datasets when applied to PCPrior for the purpose of test prioritization, the effectiveness of PCPrior (which utilizes the LightGBM model) is higher than that of PCPrior^T (which utilize the TabNet model). We can see that, in Table 8, PCPrior's APFD ranges from 0.781 to 0.905, while PCPrior^T's APFD ranges from 0.701 to 0.894. This suggests that, compared to TabNet, LightGBM performs better in leveraging the features (generated by PCPrior) for test prioritization. Some potential reasons include: 1) Different datasets and their distributions can impact the training of classification models, thereby affecting their performance; 2) The size of

Table 8. Effectiveness comparison among PCPrior and PCPrior Variants in terms of the APFD values on natural datasets

Approach	ModelNet					S3DIS					ShapeNet				
	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet
PCPrior^L	0.792	0.766	0.781	0.766	0.756	0.732	0.709	0.710	0.707	0.666	0.855	0.789	0.841	0.849	0.804
PCPrior^X	0.802	0.778	0.804	0.791	0.792	0.832	0.821	0.818	0.817	0.815	0.910	0.856	0.896	0.907	0.892
PCPrior^R	0.790	0.765	0.794	0.773	0.769	0.781	0.791	0.758	0.773	0.775	0.883	0.817	0.868	0.878	0.865
PCPrior^D	0.793	0.769	0.791	0.779	0.767	0.748	0.744	0.740	0.741	0.723	0.871	0.831	0.871	0.877	0.858
PCPrior^T	0.779	0.758	0.765	0.778	0.766	0.739	0.728	0.724	0.724	0.701	0.899	0.850	0.890	0.894	0.885
PCPrior	0.807	0.781	0.809	0.796	0.793	0.833	0.827	0.815	0.820	0.817	0.905	0.852	0.897	0.904	0.891

the dataset can also influence the model's performance. The experimental results demonstrate that LightGBM is more suitable and compatible with the feature dataset constructed by PCPrior.

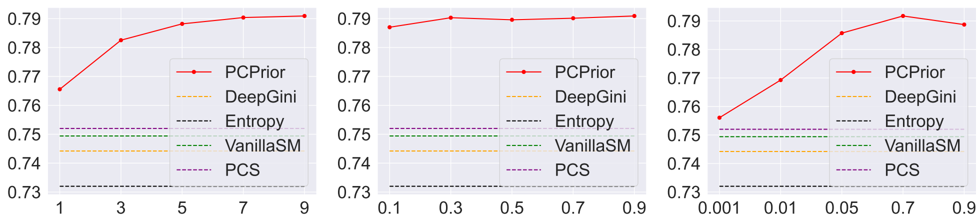
Table 9. Average comparison results among PCPrior and PCPrior Variants in terms of the PFD values on natural datasets

Data	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50	PFD-60	PFD-70
ModelNet	PCPrior^L	0.283	0.515	0.707	0.832	0.913	0.964	0.985
	PCPrior^X	0.304	0.554	0.744	0.876	0.949	0.981	0.992
	PCPrior^R	0.289	0.530	0.716	0.842	0.920	0.968	0.990
	PCPrior^D	0.295	0.541	0.719	0.847	0.927	0.967	0.985
	PCPrior^T	0.283	0.524	0.706	0.831	0.903	0.950	0.981
	PCPrior	0.305	0.567	0.760	0.882	0.950	0.983	0.994
S3DIS	PCPrior^L	0.226	0.424	0.578	0.712	0.811	0.885	0.932
	PCPrior^X	0.335	0.619	0.831	0.934	0.975	0.988	0.996
	PCPrior^R	0.285	0.524	0.709	0.841	0.922	0.969	0.990
	PCPrior^D	0.262	0.467	0.634	0.770	0.857	0.916	0.959
	PCPrior^T	0.243	0.440	0.612	0.743	0.835	0.903	0.952
	PCPrior	0.341	0.629	0.829	0.931	0.972	0.989	0.995
ShapeNet	PCPrior^L	0.427	0.690	0.832	0.907	0.944	0.967	0.980
	PCPrior^X	0.561	0.871	0.964	0.987	0.993	0.995	0.997
	PCPrior^R	0.485	0.767	0.899	0.955	0.981	0.991	0.995
	PCPrior^D	0.476	0.776	0.905	0.954	0.975	0.983	0.990
	PCPrior^T	0.543	0.843	0.948	0.976	0.986	0.992	0.995
	PCPrior	0.555	0.865	0.961	0.984	0.992	0.996	0.998

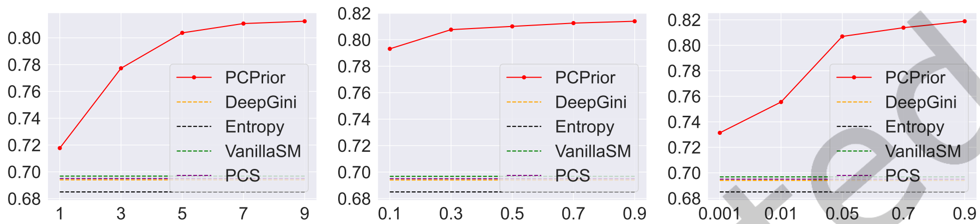
Answer to RQ2: *PCPrior and PCPrior^X exhibits better effectiveness in test prioritization compared to other PCPrior variants, thereby suggesting that the ranking model employed by PCPrior and PCPrior^X, namely LightGBM, and XGBoost, can better utilize the generated features of test inputs for test prioritization.*

5.3 RQ3: Impact of Main Parameters in PCPrior

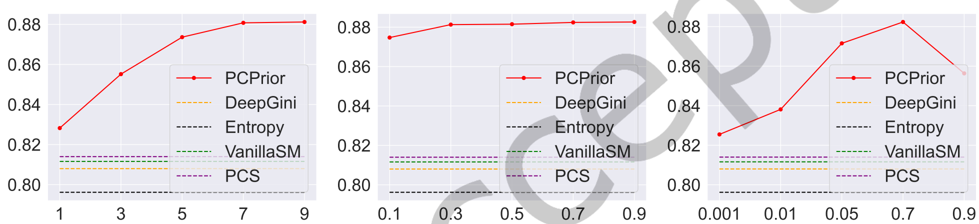
Objectives: We investigate the impact of main parameters on the effectiveness of PCPrior for test prioritization.
Experimental design: Building upon the parameter selection and consideration of parameter values in previous research [89], we conducted a systematic investigation to analyze the impact of key parameters in PCPrior. Specifically, we focused on three parameters: *max_depth* (representing the maximum tree depth for each LightGBM model), *colsample_bytree* (indicating the sampling ratio of feature columns when constructing each tree), and *learning_rate* (referring to the boosting learning rate) in the LightGBM ranking algorithm. For our investigation, we performed experiments on all subjects within the natural dataset. By observing the performance variations of PCPrior as these parameters changed, we aimed to gain insights into the influence of parameters on the effectiveness of PCPrior.
Results: The experimental results of RQ3 are presented in Figure 4, showcasing the effectiveness of PCPrior under diverse parameter settings based on average APFD values across the 15 subjects. The solid red line represents PCPrior, while the dashed lines depict the comparative methods. The findings demonstrate that PCPrior consistently outperforms all the test prioritization methods across various parameter configurations, as



(a) ModelNet, max_depth (b) ModelNet, colsample_bytree (c) ModelNet, learning_rate



(d) S3DIS, max_depth (e) S3DIS, colsample_bytree (f) S3DIS, learning_rate



(g) ShapeNet, max_depth (h) ShapeNet, colsample_bytree (i) ShapeNet, learning_rate

Fig. 4. Impact of main parameters in PCPrior

evident from the visual analysis of Figure 4. Furthermore, it can be observed that the parameter *colsample_bytree*, which determines the sampling ratio of feature columns during the construction of each tree, has a relatively modest impact on the effectiveness of PCPrior. PCPrior exhibits relative stability when this parameter is adjusted. Conversely, the parameters *max_depth* (representing the maximum tree depth for each LightGBM model) and *learning_rate* (referring to the boosting learning rate) have a relatively larger influence on the effectiveness of PCPrior. Remarkably, regardless of the extent to which the parameters influence PCPrior's effectiveness, we see that PCPrior can consistently outperform all the compared methods across different parameter settings.

> **Answer to RQ3:** *PCPrior consistently outperforms other test prioritization methods across various parameter settings. The parameter colsample_bytree has a minor impact on PCPrior's effectiveness, while the parameters max_depth and learning_rate have a relatively larger impact. However, despite these fluctuations, PCPrior consistently remains more effective than the comparative methods.*

5.4 RQ4: Effectiveness on Noisy Test Inputs

Objectives: We further investigate the effectiveness of PCPrior and its variants on noisy data.
Experimental design: In the initial phase, we introduce noise to the original 3D point cloud datasets, namely ModelNet40, ShapeNet, and S3DIS, to create noisy data. To generate a noisy dataset from an initial test set denoted as T, each test instance $t \in T$ undergoes a specific modification. Specifically, within each test instance t (a point cloud), approximately 30% of the points are subjected to a random offset in the x, y, and z coordinates, while

Table 10. Effectiveness comparison among PCPrior and the compared approaches in terms of the average APFD values on noisy datasets

Approach	ModelNet					S3DIS					ShapeNet				
	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet	DGCNN	PointConv	MSG	SSG	PointNet
Random	0.501	0.501	0.501	0.499	0.502	0.501	0.503	0.500	0.501	0.499	0.499	0.499	0.499	0.499	0.499
DeepGini	0.743	0.695	0.700	0.679	0.708	0.587	0.542	0.555	0.533	0.592	0.752	0.641	0.677	0.718	0.642
VanillaSM	0.750	0.698	0.705	0.686	0.712	0.588	0.542	0.555	0.533	0.594	0.758	0.644	0.685	0.723	0.647
PCS	0.754	0.698	0.707	0.688	0.714	0.585	0.541	0.551	0.531	0.594	0.762	0.643	0.693	0.728	0.650
Entropy	0.728	0.685	0.688	0.666	0.697	0.582	0.540	0.553	0.532	0.587	0.735	0.633	0.661	0.704	0.633
$PCPrior^L$	0.782	0.745	0.732	0.717	0.728	0.610	0.568	0.597	0.617	0.636	0.785	0.737	0.794	0.824	0.670
$PCPrior^X$	0.793	0.761	0.767	0.754	0.765	0.726	0.667	0.687	0.663	0.751	0.864	0.774	0.838	0.856	0.787
$PCPrior^R$	0.779	0.742	0.741	0.725	0.739	0.693	0.655	0.676	0.654	0.725	0.825	0.750	0.822	0.838	0.764
$PCPrior^D$	0.782	0.748	0.747	0.727	0.741	0.647	0.633	0.651	0.639	0.672	0.838	0.765	0.821	0.839	0.773
$PCPrior^T$	0.766	0.739	0.746	0.730	0.743	0.654	0.637	0.659	0.641	0.694	0.856	0.772	0.830	0.848	0.785
PCPrior	0.794	0.762	0.770	0.755	0.766	0.728	0.668	0.690	0.665	0.753	0.862	0.776	0.837	0.855	0.788

the remaining 70% of the points remain unaltered. The decision to select 30% of the points in a point cloud for displacement is because: if a large number of the points were to be shifted, it would lead to a significant number of tests being misclassified by the original model. In such a scenario, all test prioritization methods could identify a large number of misclassified tests. This, in turn, could affect the evaluation of PCPrior. Therefore, we opted to carefully select the modification ratio that is not excessively high for the evaluation of PCPrior. As a result, we generate ten noisy datasets for each original dataset, resulting in a total of 30 (3×10) noisy datasets. Each of these noisy datasets is paired with five different models, resulting in a total of 150 (30×5) subjects. Finally, we compared the effectiveness of PCPrior, its variants, and all the comparative test prioritization approaches on the generated 150 noisy subjects. On the generated noise subjects, we assessed the effectiveness of PCPrior, the confidence-based test prioritization methods, along with PCPrior variants that employed Logistic Regression [81], XGBoost [17], Random Forest [9], DNNs [82], and TabNet [3] as ranking models, respectively. We also included random selection as a baseline for comparison.

Statistical analysis Similar to RQ1, due to the inherent randomness in the model training process, we performed the experiments ten times and conducted a statistical analysis. Like in RQ1, the statistical analysis method we used is the paired two-sample t-test [46]. We calculated the p-value and effect size for the experimental results. We consider that if the p-value is less than 10^{-05}, the difference between the two sets of data is statistically significant [58]. Moreover, to ensure that the difference between the results of PCPrior and the compared approach is non-negligible, the effect size should be greater than or equal to 0.2.

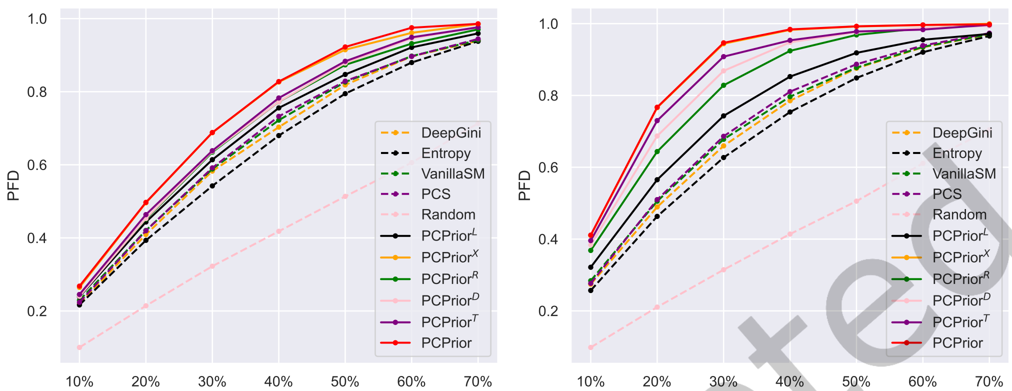
Results: The experimental results for RQ4 are presented in Table 10, Table 11, Table 12, Table 13, Table 14, and Figure 5. Specifically, Table 10 and Table 11 provide a comparative analysis of the effectiveness of PCPrior (including its variants) and various test prioritization methods in the context of noisy data, using the APFD metric. On the other hand, Table 13 and Table 14 present the comparative evaluation based on the PFD metric.

Table 11. Performance improvement of PCPrior over the compared approaches in terms of APFD on 150 noisy subjects

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.500	53.00
DeepGini	0	0.651	17.51
VanillaSM	0	0.655	16.79
PCS	0	0.656	16.62
Entropy	0	0.642	19.16
$PCPrior^L$	0	0.703	-
$PCPrior^X$	35	0.763	-
$PCPrior^R$	0	0.742	-
$PCPrior^D$	0	0.735	-
$PCPrior^T$	0	0.740	-
PCPrior	115	0.765	-

Table 12. Statistical analysis on noisy test inputs (in terms of p-value and effect size)

	Random	DeepGini	VanillaSM	PCS	Entropy
PCPrior (p-value)	1.156×10^{-10}	1.688×10^{-08}	3.521×10^{-08}	5.049×10^{-08}	3.385×10^{-09}
PCPrior (effect size)	4.329	2.958	2.792	2.713	3.352



(a) ModelNet (noisy), PointNet (b) ShapeNet (noisy), DGCNN

Fig. 5. Test prioritization effectiveness among PCPrior and the compared approaches for ModelNet(Noisy) with PointNet and ShapeNet(Noisy) with DGCNN on noisy datasets. X-Axis: the percentage of prioritized tests; Y-Axis: the percentage of detected misclassified tests.

Table 10 shows the comparison results of PCPrior, its variants, and comparative methods on noisy test inputs in terms of APFD. We found that the effectiveness of PCPrior and its variants surpasses that of all compared test prioritization methods in each case. Specifically, the APFD values for PCPrior range from 0.665 to 0.862. For PCPrior's variants, the APFD values range from 0.568 to 0.864. For the compared test prioritization methods, the APFD values range from 0.499 to 0.762. Furthermore, Table 11 provides a more detailed analysis by presenting the number of cases in which each test prioritization method performs the best, the average APFD value, and the improvement of PCPrior relative to each comparative method. We see that, on noisy test inputs, PCPrior's average APFD is 0.765, while the range for its variants is 0.703 to 0.763. The average APFD range for the benchmark methods is 0.500 to 0.656. Notably, PCPrior performs the best in 76.7% (115 out of 150) of the cases, while PCPrior^X performs the best in 23.3% (35 out of 150) of the cases. PCPrior continues to outperform the variants of PCPrior that utilize DNN ranking models (PCPrior^T and PCPrior^N) in all cases. Moreover, PCPrior shows an improvement ranging from 16.62% to 53.00% over all the comparative methods. The above experimental results demonstrate that, under the APFD measurement, the average effectiveness of PCPrior surpasses all its variants and comparative methods on noisy datasets.

The results from the statistical analysis on noisy test inputs are presented in Table 12. We see that the p-values for the experimental results of PCPrior and each of the compared methods are all less than 10^{-05}, indicating that PCPrior statistically outperforms all the test prioritization methods on noisy datasets. For instance, the p-value between PCPrior and DeepGini is 1.688×10^{-08}. The p-value between PCPrior and PCS is 5.049×10^{-08}. Furthermore, all the effect sizes of PCPrior and the compared approaches exceed 0.2, demonstrating a non-negligible difference. Notably, all the effect sizes are greater than 0.8. For example, the effect size between PCPrior and VanillaSM is 2.792, and the effect size between PCPrior and Entropy is 3.352. According to Cohen's d [42], this implies that the difference in experimental results between PCPrior and the compared methods is not only statistically significant but also relatively "large" in scale.

Table 13. Effectiveness comparison of PCPrior and the compared approaches in terms of the PFD values on noisy datasets

Data	Approach	#Best cases in PFD				Average PFD			
		PFD-10	PFD-20	PFD-30	PFD-40	PFD-10	PFD-20	PFD-30	PFD-40
ModelNet	Random	0	0	0	0	0.099	0.201	0.300	0.402
	DeepGini	0	0	0	0	0.219	0.404	0.564	0.701
	VanillaSM	0	0	0	0	0.225	0.417	0.577	0.712
	PCS	0	0	0	0	0.220	0.416	0.583	0.719
	Entropy	0	0	0	0	0.210	0.387	0.542	0.677
	$PCPrior^L$	1	0	0	0	0.246	0.459	0.635	0.772
	$PCPrior^X$	19	17	20	13	0.264	0.494	0.687	0.830
	$PCPrior^R$	1	1	0	0	0.252	0.465	0.641	0.777
	$PCPrior^D$	0	0	0	0	0.250	0.468	0.647	0.787
	$PCPrior^T$	0	0	0	0	0.252	0.470	0.651	0.790
	PCPrior	29	32	30	37	0.265	0.497	0.689	0.833
S3DIS	Random	0	0	0	0	0.099	0.201	0.302	0.402
	DeepGini	0	0	0	0	0.133	0.258	0.375	0.486
	VanillaSM	0	0	0	0	0.135	0.260	0.377	0.489
	PCS	0	0	0	0	0.130	0.255	0.373	0.485
	Entropy	0	0	0	0	0.131	0.254	0.370	0.480
	$PCPrior^L$	0	0	0	0	0.151	0.291	0.419	0.541
	$PCPrior^X$	8	8	6	7	0.188	0.369	0.536	0.689
	$PCPrior^R$	0	0	0	0	0.182	0.350	0.509	0.654
	$PCPrior^D$	0	0	0	0	0.173	0.331	0.476	0.608
	$PCPrior^T$	0	0	0	0	0.176	0.339	0.488	0.623
	PCPrior	42	42	44	43	0.189	0.370	0.539	0.692
ShapeNet	Random	0	0	0	0	0.099	0.199	0.298	0.399
	DeepGini	0	0	0	0	0.212	0.390	0.544	0.675
	VanillaSM	0	0	0	0	0.221	0.403	0.557	0.685
	PCS	0	0	0	0	0.221	0.411	0.568	0.694
	Entropy	0	0	0	0	0.201	0.372	0.519	0.649
	$PCPrior^L$	0	0	0	0	0.277	0.519	0.703	0.822
	$PCPrior^X$	17	24	39	30	0.317	0.616	0.841	0.951
	$PCPrior^R$	0	0	0	0	0.303	0.567	0.769	0.894
	$PCPrior^D$	0	0	0	0	0.308	0.585	0.792	0.912
	$PCPrior^T$	0	0	0	0	0.314	0.606	0.826	0.939
	PCPrior	33	26	11	20	0.318	0.614	0.838	0.949

Table 14. Average effectiveness comparison of PCPrior and the compared approaches in terms of the PFD values on noisy datasets

Approach	#Best cases in PFD				Average PFD			
	PFD-10	PFD-20	PFD-30	PFD-40	PFD-10	PFD-20	PFD-30	PFD-40
Random	0	0	0	0	0.099	0.201	0.300	0.401
DeepGini	0	0	0	0	0.188	0.351	0.494	0.621
VanillaSM	0	0	0	0	0.194	0.36	0.504	0.629
PCS	0	0	0	0	0.190	0.361	0.508	0.633
Entropy	0	0	0	0	0.181	0.337	0.477	0.602
$PCPrior^L$	1	0	0	0	0.225	0.423	0.585	0.712
$PCPrior^X$	44	49	65	50	0.256	0.493	0.688	0.823
$PCPrior^R$	1	1	0	0	0.246	0.461	0.639	0.775
$PCPrior^D$	0	0	0	0	0.244	0.461	0.639	0.769
$PCPrior^T$	0	0	0	0	0.248	0.472	0.655	0.784
PCPrior	104	100	85	100	0.257	0.494	0.689	0.825

Table 13 and Table 14 present a comparative analysis regarding the PFD metric. It is observed that in Table 13, the best performance is consistently achieved by PCPrior or its variants across all cases. Table 14 provides a deeper analysis of this finding. When considering different percentages of test data prioritization, PCPrior consistently outperforms other approaches in terms of effectiveness, as evidenced by the highest number of best-performing cases and the highest average PFD values. Figure 5 visually demonstrates the performance comparison of PCPrior, its variants, and the comparative methods on noisy data. The solid lines depict PCPrior and its variant methods, while the dashed lines represent the comparative methods. We see that across the noisy dataset, PCPrior and all its variants exhibit higher effectiveness compared to all comparative methods. Furthermore, PCPrior demonstrates superior performance when compared to its variants.

> **Answer to RQ4:** *PCPrior consistently exhibits superior performance in comparison to all the test prioritization approaches considered in the context of noisy data, as evaluated by APFD and PFD. Notably, the average improvement achieved in terms of APFD ranges from 16.62% to 53.00%, highlighting the significant effectiveness of PCPrior over the compared methods. Furthermore, PCPrior consistently outperforms its variants in a majority of cases.*

5.5 RQ5: Feature contribution analysis

Objectives: We investigate the contributions of each type of features on the effectiveness of PCPrior for test prioritization. Our investigation revolves around two primary sub-questions, as outlined below:

- **RQ-5.1** Based on the ablation study, to what extent does each type of features contribute to the effectiveness of PCPrior?
- **RQ-5.2** What is the distribution of feature types among the top-N most contributing features towards PCPrior?

Experimental design: We conduct two experiments below to answer the above two sub-questions.
[Experiment ❶] In the original PCPrior framework, a comprehensive set of four feature types is generated, namely mutation features (MF), spatial features (SF), uncertainty features (UF), and prediction features (PF). To compare the contributions of each feature type on PCPrior's effectiveness, we conducted a carefully designed ablation study following the prior work [25]. More specifically, we individually removed one type of features and evaluated PCPrior's effectiveness under these modified conditions. For instance, to assess the contribution of SF features, PCPrior is executed with SF features excluded while retaining the other three feature types. The resulting performance of PCPrior is then evaluated under these adjusted circumstances. Similarly, to gauge the contribution of MF features, PCPrior is executed without generating MF features while keeping generating the other three feature types. The performance of PCPrior is subsequently assessed in this context. By conducting the aforementioned ablation study, we can determine the contribution of each feature type to the overall effectiveness of PCPrior.
[Experiment ❷] The method we employed to evaluate the contributions of features is the cover metric within the XGBoost algorithm [17]. Initially, we utilized the cover metric to compute the importance scores of each feature used by PCPrior for test prioritization. Subsequently, we selected the top-N most important features based on these scores. By analyzing the categorization of these features, we investigated the contributions of different feature types to the effectiveness of PCPrior. Below, we provide an overview of how XGBoost quantifies feature importance.

The cover metric employed in XGBoost serves as a means to quantify the importance of features by assessing the average coverage of individual instances across the leaf nodes within a decision tree. This metric operates by evaluating the frequency with which a specific feature is utilized for partitioning the data across the entirety of the ensemble's trees. The coverage values associated with each feature across all trees are subsequently aggregated, resulting in a cumulative coverage value. To obtain the average coverage of each instance by the leaf nodes, the cumulative coverage value is normalized in relation to the total number of instances. Consequently, the

Table 15. Ablation study on different features of PCPrior: Mutation Features(MF), Spatial Features(SF), Uncertainty Features(UF), Prediction Features(PF). 'w/o' means 'without'

Approach	Dataset			Average
	ModelNet	S3DIS	ShapeNet	
PCPrior w/o MF	0.788	0.811	0.875	0.825
PCPrior w/o SF	0.769	0.699	0.841	0.769
PCPrior w/o UF	0.785	0.816	0.871	0.824
PCPrior w/o PF	0.782	0.778	0.874	0.811
PCPrior	0.797	0.822	0.890	0.836

derived coverage value of a given feature plays a crucial role in determining its significance, with features that demonstrate higher coverage values being considered more important.

Results: The experimental results of RQ5.1 are presented in Table 15. In this table, 'w/o' stands for 'without.' For example, 'PCPrior w/o SF' refers to executing PCPrior without generating the spatial features. From Table 15, we see that the original PCPrior achieves the highest average effectiveness. Removing any type of feature results in a decrease in the effectiveness of PCPrior, demonstrating that each type of features contributes to PCPrior's effectiveness. For instance, on the Modelnet dataset, the average APFD value of the original PCPrior is 0.797. Removing spatial features results in a decline of PCPrior's average APFD to 0.769, while the removal of mutation features causes a decrease to 0.788, uncertainty features to 0.785, and prediction features to 0.782.

Furthermore, among all four types of features, spatial features demonstrate the highest average contributions. This inference is drawn from the following findings: When removing spatial features, PCPrior's effectiveness shows the largest average decrease. Specifically, when removing spatial features (SF features), the average APFD decreases by 0.067. In comparison, the removal of mutation features (MF) leads to an average APFD decrease of 0.011, uncertainty features (UF) result in an average APFD decrease of 0.012, and prediction features (PF) show an average APFD decrease of 0.025. Moreover, across all datasets, removing spatial features results in the highest average decrease in PCPrior's effectiveness.

Answer to RQ5.1: *The ablation study demonstrates that each type of features contributes to the effectiveness of PCPrior. Moreover, spatial features show the highest average contributions.*

The findings of RQ5.2 are presented in Table 16, where the scores represent the importance levels of each feature. For each combination of model and dataset, we present the top-N features that contribute the most. It is worth noting that abbreviations SF, MF, PF, and UF are used to represent spatial features, mutation features, prediction features, and uncertainty features, respectively. Moreover, the numbers after the feature abbreviations indicate the indices of the corresponding features. For instance, *SF-23* represents the spatial feature with index 23. From Table 16, it can be observed that all four types of features consistently appear among the top-N most contributing features across various subjects. As an example, in the case of the PointConv subject with the S3DIS dataset, SF features account for 50%, UF features account for 30%, MF features account for 10%, and PF features account for 10%. Remarkably, among the 15 subjects investigated, in 93.3% (14 out of 15) of the cases, the top 10 contributing features include three or more distinct feature types. These experimental findings provide robust evidence that all three feature categories play pivotal roles in the effectiveness of PCPrior.

Answer to RQ5.2: *All four types of features, namely spatial features, mutation features, uncertainty features, and prediction features, exhibit consistent presence among the top-N most influential features across diverse subjects.*

Table 16. Top-10 most contributing features on the effectiveness of PCPrior

Data	Rank	DGCNN		PointConv		MSG		SSG		PointNet	
		Feature	Value	Feature	Value	Feature	Value	Feature	Value	Feature	Value
ModelNet	1	SF-23	348	MF-132	272	SF-f23	271	SF-46	261	MF-120	309
	2	MF-141	203	MF-126	261	PF-112	260	MF-147	238	PF-112	276
	3	MF-143	197	SF-52	243	MF-120	210	MF-114	225	MF-144	265
	4	PF-112	195	PF-112	221	MF-143	173	PF-112	224	SF-52	256
	5	MF-137	187	MF-125	219	MF-127	167	MF-131	199	SF-28	210
	6	MF-145	178	MF-139	214	SF-52	163	MF-127	196	PF-90	174
	7	SF-22	171	MF-129	213	SF-28	160	MF-122	155	SF-69	167
	8	MF-123	169	MF-135	207	MF-135	141	SF-42	142	SF-1	166
	9	MF-131	167	UF-118	206	SF-f22	138	SF-29	131	MF-139	165
	10	MF-136	155	MF-124	201	PF-88	134	SF-25	128	SF-25	164
S3DIS	1	UF-85	166	SF-18	168	UF-85	175	UF-87	190	UF-87	156
	2	PF-75	112	UF-85	140	UF-87	151	SF-55	161	UF-85	119
	3	SF-61	111	SF-61	127	UF-86	115	UF-86	148	SF-65	115
	4	SF-60	106	UF-87	123	SF-28	114	SF-19	128	SF-68	106
	5	SF-62	105	MF-106	100	MF-91	111	SF-20	110	PF-74	105
	6	SF-2	94	PF-80	99	PF-73	110	UF-90	107	PF-82	94
	7	UF-86	93	SF-60	99	SF-62	107	PF-84	102	SF-17	87
	8	PF-74	86	SF-62	93	SF-16	102	SF-65	101	UF-90	86
	9	SF-35	83	SF-4	92	MF-95	98	MF-99	99	PF-79	86
	10	SF-16	82	UF-86	90	PF-74	96	PF-78	98	PF-80	85
ShapeNet	1	PF-122	908	MF-135	1223	UF-122	952	UF-122	984	UF-122	1021
	2	MF-151	554	MF-141	1059	UF-124	518	MF-156	536	PF-101	502
	3	MF-148	542	UF-130	934	MF-155	505	UF-124	527	PF-100	435
	4	MF-140	539	MF-153	850	MF-145	503	SF-70	496	PF-110	523
	5	MF-145	498	MF-143	801	MF-136	475	PF-107	477	PF-94	415
	6	SF-42	486	PF-122	798	PF-121	467	SF-42	476	SF-71	398
	7	PF-107	486	MF-154	776	MF-153	460	MF-149	435	UF-124	391
	8	PF-116	424	MF-150	749	UF-126	446	MF-155	408	SF-42	389
	9	PF-109	423	MF-148	725	SF-42	446	PF-104	405	PF-87	383
	10	PF-110	376	PF-121	701	SF-71	429	MF-131	404	PF-90	371

5.6 RQ6: Retraining 3D shape classification models with PCPrior and uncertainty-based methods

Objectives: We investigate whether PCPrior and uncertainty-based test prioritization approaches are effective in selecting informative retraining inputs to enhance the performance of a 3D shape classification model.

Experimental design: Building on the previous research [58], we structured our retraining experiments in the following manner. First, we randomly divided the point cloud dataset into three parts: the training set, the candidate set, and the test set, in a 4:4:2 ratio. The candidate set was used for retraining, while the test set was reserved for evaluation purposes and remained untouched. In the first phase, we trained a 3D shape classification model using only the initial training set. In the second round, we integrate an extra 10% of new inputs from the candidate set into the current training set without replacement. The chosen inputs for inclusion are those prioritized in the top 10% by PCPrior and the compared test prioritization approaches. The prioritization range we selected for retraining is from 10% to 70%. We chose this range because, according to the experimental results (cf. Section 5.1), when prioritizing up to 70%, PCPrior can identify the majority of misclassified inputs in the dataset (99.6%), as indicated in Table 6. For example, in the ShapeNet dataset, within the 70% prioritized test set, PCPrior has identified 99.8% of misclassified inputs. Given that the primary objective of this research question is to validate PCPrior's effectiveness in retraining, we chose a retraining range of up to 70%. Following prior work [58], we retrained the model using the expanded training set, ensuring equal treatment of both old and new training data. This retraining was repeated in five rounds. The reason for opting to conduct retraining five times is that the training process of DNN models involves various random factors, and conducting multiple rounds of retraining can contribute to ensuring the stability and reproducibility of the results. On the other hand, excessive retraining can lead the model to over-optimize for a specific dataset, resulting in overfitting. Therefore, based on

Table 17. The average accuracy value after retraining with 10%~70% prioritized tests

Approach	Accuracy of percentage of datasets							Average
	10%	20%	30%	40%	50%	60%	70%	
Random	0.847	0.855	0.865	0.872	0.879	0.886	0.887	0.870
DeepGini	0.846	0.866	0.872	0.880	0.889	0.896	0.898	0.878
VanillaSM	0.850	0.867	0.870	0.881	0.890	0.891	0.898	0.878
PCS	0.846	0.861	0.868	0.884	0.888	0.893	0.898	0.877
Entropy	0.846	0.861	0.869	0.881	0.886	0.895	0.896	0.876
PCPrior	0.851	0.868	0.873	0.883	0.888	0.898	0.901	0.880

the experimental experience of existing studies [32], we choose to conduct five rounds of retraining. To account for the inherent randomness in model training, we repeated all experiments three times and reported the average results across these repetitions.

Results: The experimental results for RQ6 are presented in Table 17, which illustrates the average accuracy of 3D shape classification models after retraining. In each case, we have highlighted the approach with the highest effectiveness in grey for a quick and straightforward interpretation of the findings. As shown in Table 17, PCPrior and all uncertainty-based approaches demonstrate better average effectiveness compared to random selection. However, the improvements they achieved are relatively small. For instance, when selecting 10% of tests for retraining the original model, PCPrior's selected samples result in a post-retrain model accuracy of 0.851, while uncertainty-based methods range from 0.846 to 0.850. In contrast, the random selection yields an accuracy of 0.847. Similarly, when choosing 70% of tests for retraining the original model, PCPrior's selected samples result in a post-retrain model accuracy of 0.901, while uncertainty-based methods range from 0.896 to 0.898. In comparison, the random selection yields an accuracy of 0.887.

The reasons for the aforementioned findings, where PCPrior and uncertainty-based methods show only small improvements over random selection in enhancing model accuracy, include:

- **Lack of Diversity**: PCPrior and uncertainty-based methods focus on identifying corner cases, which are tests that the model finds more challenging. Consequently, the tests identified can lack diversity. In contrast, random selection provides a broader and more diverse set of samples, contributing to the model learning more comprehensive data features and thereby improving its generalization capability.
- **Overfitting Risk**: Concentrating on samples the model is most likely to predict incorrectly can lead to overfitting. These samples can exhibit certain extreme or uncommon features, causing the model to overly adapt to these specific cases after retraining and ignoring more widespread patterns.

Moreover, another observation from the results in Table 17 is that PCPrior performs better than uncertainty-based methods on average. Specifically, PCPrior performs the best in 75% (6 out of 8) cases, while uncertainty-based methods perform the best in only 25% (2 out of 8) cases. Moreover, after retraining the original model with tests selected by PCPrior, the average accuracy of the resulting model is 0.880. In contrast, for uncertainty-based methods, the range is from 0.876 to 0.878.

Answer to RQ6: *PCPrior and uncertainty-based methods perform better than the random selection approach. However, the improvement achieved is relatively modest, suggesting that these prioritization approaches, aimed at identifying potentially misclassified tests, can guide the retraining of 3D shape classification models but with limited effectiveness. Additionally, PCPrior demonstrates better effectiveness compared to uncertainty-based test prioritization methods.*

6 DISCUSSION

6.1 Limitations of PCPrior

PCPrior suffers from a notable limitation regarding its ability to ensure the diversity of the selected data, which has also been recognized in previous investigations on uncertainty-based test prioritization techniques [28]. This concern arises from the fact that neither PCPrior nor these earlier approaches account for diversity during the process of prioritizing test inputs. However, despite this shared limitation, PCPrior has demonstrated considerable effectiveness in identifying a substantial majority of misclassified test inputs by leveraging a small proportion of prioritized test cases. The experimental results illustrate that PCPrior can detect over 95% of misclassified tests on natural datasets by prioritizing a mere 50% of the test inputs. This noteworthy performance highlights PCPrior's ability to efficiently identify a significant proportion of misclassified tests using a reduced set of prioritized tests, even without explicitly ensuring the diversity. While prioritizing diverse misclassified tests undoubtedly enhances overall testing quality, in practical scenarios with limited time and resource constraints, prioritizing a significant majority of misclassified tests can still be a viable strategy. Therefore, the capacity of PCPrior to identify a significant proportion of misclassified tests while operating within the constraints of a reduced number of prioritized tests becomes particularly advantageous in situations where time and resources are scarce.

Another limitation is that PCPrior is specifically designed for classification models and cannot be adapted for regression models. This is primarily due to two reasons: 1) PCPrior requires generating mutation features from tests for test prioritization. However, for a given test, generating mutation features involves comparing whether the model's predictions for this test and its variants are the same. This approach is not applicable to regression models because the predictions of regression models are continuous numerical values. 2) PCPrior requires generating prediction features and uncertainty features for test prioritization. For a given test, the generation of these two types of features requires the model to predict the probabilities of this test belonging to each category. Therefore, PCPrior cannot be applied to regression models.

6.2 Generality of PCPrior

Our experimental findings have validated the effectiveness of PCPrior based on a large number of subjects, encompassing both natural and noisy scenarios. Although our study initially focused on three datasets, PCPrior can be generalized to a broader range of 3D shape classification domains. The adaptability of PCPrior stems from its core process, which is the generation of four types of features: spatial features, mutation features, prediction features, and uncertainty features. PCPrior can perform test prioritization through an automated pipeline when the evaluated model and dataset meet the criteria for generating these four types of features. Below, we provide a detailed explanation of the specific conditions that the evaluated model and dataset require to meet in order to utilize PCPrior:

- **Requirement 1: Point Cloud Dataset.** The generation of spatial features and mutation features requires the dataset to be a point cloud dataset. This is because these two features are specifically tailored for point cloud data. For a given point cloud dataset, PCPrior can automatically generate its spatial feature and mutation features.
- **Requirement 2: Classification Tasks.** The generation of the prediction features and uncertainty features necessitates that both the model and the dataset be oriented toward classification tasks. This is because these two types of features are generated from the model's predictions for each test within the test set. Specifically, for a given test, the generation of these two types of features requires the model to predict the probabilities of this test belonging to each category.

Models and datasets that meet the above conditions can use PCPrior for test prioritization, making PCPrior widely applicable in a diverse range of 3D shape classification tasks.

6.3 Threats to Validity

6.3.1 *Internal Threats to Validity.*

Internal threats to validity primarily arise from the implementation of our proposed PCPrior methodology and the compared approaches. To address these threats, we implemented PCPrior using the widely adopted PyTorch library. Additionally, we utilized the original implementations of the compared approaches as provided by their respective authors, minimizing potential implementation biases. Another internal threat emerges from the inherent randomness associated with model training. To mitigate this threat and ensure the stability of our experimental results, we conducted a statistical analysis. Specifically, we performed ten repetitions of the training process and calculated the statistical significance of the experimental results, thereby reducing the influence of randomness.

6.3.2 *External Threats to Validity.*

External threats to validity primarily reside in the 3D point cloud dataset and DNN models employed in our study. To mitigate these threats, we adopted a large number of subjects, encompassing both natural and noisy data, thus ensuring a comprehensive exploration of various scenarios. By including diverse data types, we aimed to enhance the robustness and generalizability of our findings. As a future direction, we aim to extend the application of PCPrior to 3D point cloud datasets characterized by diverse properties, thereby broadening the scope and applicability of our proposed methodology.

7 RELATED WORK

7.1 Test Prioritization Techniques

Test prioritization aims to determine the optimal order for executing test cases, thereby enabling the early detection of system bugs. The idea was first mentioned by Wong et al. [91]. In field of conventional software engineering [11–13, 22, 31], several corresponding studies have been conducted. Di Nardo *et al.* [22] conducted a study evaluating the effectiveness of coverage-based prioritization strategies using real-world regression faults. Their research shed light on the efficiency of different techniques in detecting bugs. Henard *et al.* [31] conducted a comprehensive investigation to compare existing test prioritization approaches, specifically focusing on white-box and black-box strategies. Their findings revealed minimal distinctions between these two categories of strategies. Chen *et al.* [13] proposed the LET (Learning-based and Execution Time-aware Test prioritization) technique for prioritizing test programs in compiler testing, demonstrating its effectiveness. LET employs a learning process to identify program features and predict the bug-revealing probability of new test programs, along with a scheduling process that prioritizes test programs based on their bug-revealing probabilities.

Furthermore, several studies have focused on addressing the test prioritization problem using mutation testing techniques [20, 39, 54, 66, 79]. Shin *et al.*[79] proposed a diversity-aware mutation adequacy criterion to guide test case prioritization and empirically evaluated mutation-based prioritization techniques using large-scale developer-written test cases. Papadakis *et al.*[66] introduced the concept of mutating Combinatorial Interaction Testing models and prioritizing tests based on their ability to detect mutants. They demonstrated a strong correlation between the number of model-based mutants killed and code-level faults detected by the test cases.

Regarding test prioritization for DNNs, Feng *et al.*[28] proposed DeepGini, which identifies possibly misclassified tests based on model uncertainty. DeepGini assumes that a test is more likely to be mispredicted if the DNN outputs similar probabilities for each class. Weiss *et al.*[90] conducted a comprehensive investigation of various DNN test input prioritization techniques, including several uncertainty-based metrics such as Vanilla Softmax, Prediction-Confidence Score (PCS), and Entropy. Moreover, Wang *et al.* [89] developed PRIMA, an intelligent mutation analysis-based approach, specifically tailored for prioritizing test inputs in DNNs. However, the mutation rules of PRIMA are not adapted to handle 3D point data, which constitutes unstructured sets of points in three-dimensional space. To address this limitation, we propose PCPrior, a novel test prioritization technique that is specifically designed for 3D point cloud data. PCPrior effectively generates a set of features to facilitate test prioritization.

7.2 Mutation Testing for DNNs

In the field of mutation testing for DNNs, various studies [34, 37, 39, 57, 78] have been conducted, focusing on the development of different mutation operators and frameworks. Shen *et al.* introduced MuNN [78], a mutation analysis method specifically designed for neural networks. MuNN defined five mutation operators based on the characteristics of neural networks. The research findings highlighted that mutation analysis exhibited strong domain-specific characteristics, indicating the necessity of domain-specific mutation operators to enhance the analysis process. Ma *et al.* [57] proposed DeepMutation, a methodology for assessing the quality of test data in DL systems using mutation testing. They devised a collection of source-level and model-level mutation operators to introduce faults into the training data, training programs, and DL models. Subsequently, Hu *et al.* [34] extended DeepMutation to DeepMutation++ by introducing a new set of mutation operators for feed-forward neural networks (FNNs) and Recurrent Neural Networks (RNNs) and enabled dynamic mutation of run-time states in RNNs. Jahangirova *et al.* [39] conducted a comprehensive empirical study on the DL mutation operators in the existing literature. Their investigation shed light on the necessity for a stochastic definition of mutation killing. Furthermore, they successfully identified a subset of mutation operators that exhibit high effectiveness, along with the associated configurations that yield the highest efficacy. Humbatova *et al.* presented DeepCrime [37], the first mutation testing tool that implemented a set of DL mutation operators based on real DL faults. This tool provided a comprehensive framework for evaluating the robustness and fault tolerance of DNNs.

7.3 Deep Neural Network Testing

In addition to test input prioritization, test selection [58] is another approach for improving the efficiency of DNN testing. The goal of test selection is to estimate the accuracy of the entire set by only labeling a selected subset of test inputs, thereby reducing the labeling cost associated with DNN testing. Several effective test selection methods have been proposed in the literature [14, 29, 44, 51, 58]. Li *et al.*[51] introduced Cross Entropy-based Sampling (CES), a method for selecting a representative subset of test inputs to estimate the accuracy of the entire testing set. CES minimizes the cross-entropy between the selected set and the original test set to ensure that the distribution of the selected test set is similar to that of the original set. Chen *et al.*[14] proposed Practical Accuracy Estimation (PACE) for test selection. The basic principle of PACE involves clustering all the tests in the test set and using the MMD-critic algorithm [44] to perform prototype selection. For the remaining test inputs that do not belong to any group, adaptive random testing is employed for test selection.

In addition to focusing on improving the efficiency of DNN testing, many studies in the field of DNN testing [34, 45, 55–57, 69] concentrate on measuring the adequacy of DNNs. Pei *et al.*[69] proposed neuron coverage, a metric for evaluating how well a test set covers the logic of a DNN model. Ma *et al.*[56] introduced DeepGauge, a set of coverage criteria to measure the test adequacy of DNNs. DeepGauge considers neuron coverage as an important indicator of the effectiveness of a test input. Moreover, they proposed new metrics with different granularities based on neuron coverage to differentiate adversarial attacks from legitimate test data. Kim *et al.* [45] proposed surprise adequacy as a measure of identifying the effectiveness of a test input within a test set. Surprise adequacy focuses on measuring the surprise of a test input with respect to the training set, where surprise is defined as the difference in the activation value of neurons when faced with this new test input. Dola *et al.* [23] proposed the Input Distribution Coverage (IDC) framework to evaluate the black-box test adequacy of DNNs. The framework utilizes a Variational Autoencoder (VAE) to transform test inputs into feature vectors, establishing a coverage domain. Within this domain, Combinatorial Interaction Testing (CIT) metrics are applied to measure test coverage. Riccio *et al.* [75] introduced the notion of "mutation adequacy" to assess the effectiveness of test sets in identifying artificially injected faults (mutations) in deep learning systems. Moreover, they proposed DEEPMETIS as a solution to enhance the mutation adequacy of the test set (i.e., improving the test set's ability to detect mutations).

Furthermore, several studies focused on utilizing the decision boundary to enhance the quality assurance of DL-based software. Riccio *et al.* [76] proposed the notion of the "frontier of behaviors" referring to the inputs at which a DL system begins to exhibit misbehavior. This concept serves as a metric for evaluating the quality of DL systems. The assessment involves determining whether the frontier of misbehaviors extends beyond the system's validity domain, in which case the quality check is deemed successful. Conversely, if the frontier intersects with the validity domain, it indicates quality deficiencies in the system. Biagiola *et al.* [7] introduced an innovative approach to assessing the adaptability of reinforcement learning (RL) systems, focusing on their capacity to adjust to dynamic environments. Their method involves computing the adaptation boundary within a changing environment and presenting them through two-dimensional or multi-dimensional adaptability/anti-regression heatmaps. These visualizations serve to quantify the system's adaptability and anti-regression capabilities. Fahmy *et al.* [27] introduced Simulator-based Explanations for DNN Failures (SEDE) as a technique aimed at bolstering the quality assurance of DNNs within safety-critical systems. SEDE proficiently identifies and simulates events that trigger hazards, leading to DNN failures. This is achieved by generating images with features akin to those causing failures, which are then used for retraining, ultimately improving DNN accuracy.

8 CONCLUSION

To address the issue of high labeling costs for 3D point cloud data, we propose a novel approach called PCPrior, which aims to prioritize test inputs that are likely to be misclassified. By focusing on these challenging inputs, developers can allocate their limited labeling budgets more efficiently, ensuring that the most critical test cases are labeled first, which can lead to cost savings and a more cost-effective testing process. The core idea behind PCPrior is that test inputs closer to the decision boundary of the model are more likely to be predicted incorrectly. In order to capture the spatial relationship between a point cloud test and the decision boundary, we adopt a vectorization approach that transforms the point cloud data into a low-dimensional space, towards revealing the underlying proximity between the point cloud data and the decision boundary indirectly. To implement the vectorization strategy, we generate four distinct types of features for each point cloud (test): Spatial Features, Mutation Features, Prediction Features, and Uncertainty Features. For each test input, the four generated features are concatenated into a final feature vector. Subsequently, PCPrior employs a ranking model to automatically learn the probability of a test input being mispredicted by the model based on its final feature vector. Finally, PCPrior utilized the obtained probability values to rank all the test inputs. In order to assess the performance of PCPrior, we conducted a comprehensive evaluation involving a diverse set of 165 subjects. These subjects encompass both natural datasets and noise datasets. We compared the effectiveness of PCPrior with several established test prioritization approaches that have exhibited effectiveness in prior studies. The empirical results demonstrate the remarkable effectiveness of PCPrior. Specifically, on natural datasets, PCPrior consistently performs better than all the comparative test prioritization approaches, yielding an improvement ranging from 10.99% to 66.94% in terms of APFD. Moreover, on noisy datasets, the improvement ranges from 16.62% to 53%.

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

https://github.com/yinghuali/PCPrior

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] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*. PMLR, 40–49.

[2] Khalid M Al-Gethami, Mousa T Al-Akhras, and Mohammed Alawairdhi. 2021. Empirical evaluation of noise influence on supervised machine learning algorithms using intrusion detection datasets. *Security and Communication Networks* 2021 (2021), 1–28.

[3] Sercan Ö Arik and Tomas Pfister. 2021. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 6679–6687.

[4] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 2016. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1534–1543.

[5] Sanjib Basu and Anirban DasGupta. 1997. The mean, median, and mode of unimodal distributions: a characterization. *Theory of Probability & Its Applications* 41, 2 (1997), 210–223.

[6] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. 2020. Deep learning on 3D point clouds. *Remote Sensing* 12, 11 (2020), 1729.

[7] Matteo Biagiola and Paolo Tonella. 2022. Testing the plasticity of reinforcement learning-based systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–46.

[8] Dorit Borrmann, Andreas Nuechter, and Thomas Wiemann. 2018. Large-scale 3d point cloud processing for mixed and augmented reality. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, xxxv–xxxv.

[9] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[10] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012* (2015).

[11] Junjie Chen. 2018. Learning to accelerate compiler testing. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 472–475.

[12] Junjie Chen, Yanwei Bai, Dan Hao, Yingfei Xiong, Hongyu Zhang, and Bing Xie. 2017. Learning to prioritize test programs for compiler testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 700–711.

[13] 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.

[14] 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.

[15] 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.

[16] Siheng Chen, Baoan Liu, Chen Feng, Carlos Vallespi-Gonzalez, and Carl Wellington. 2020. 3d point cloud processing and learning for autonomous driving: Impacting map creation, localization, and perception. *IEEE Signal Processing Magazine* 38, 1 (2020), 68–86.

[17] 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.

[18] Chuang-Yuan Chiu, Michael Thelwell, Terry Senior, Simon Choppin, John Hart, and Jon Wheat. 2019. Comparison of depth cameras for three-dimensional reconstruction in medicine. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine* 233, 9 (2019), 938–947.

[19] Yaodong Cui, Ren Chen, Wenbo Chu, Long Chen, Daxin Tian, Ying Li, and Dongpu Cao. 2021. Deep learning for image and point cloud fusion in autonomous driving: A review. *IEEE Transactions on Intelligent Transportation Systems* 23, 2 (2021), 722–739.

[20] Xueqi Dang, Yinghua Li, Mike Papadakis, Jacques Klein, Tegawendé F Bissyandé, and Yves Le Traon. 2024. Test input prioritization for Machine Learning Classifiers. *IEEE Transactions on Software Engineering* (2024).

[21] 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).

[22] 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.

[23] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. 2023. Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–48.

[24] Bertrand Douillard, James Underwood, Noah Kuntz, Vsevolod Vlaskine, Alastair Quadros, Peter Morton, and Alon Frenkel. 2011. On the segmentation of 3D LIDAR point clouds. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2798–2805.

[25] 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.

[26] 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.

[27] Hazem Fahmy, Fabrizio Pastore, Lionel Briand, and Thomas Stifter. 2023. Simulator-based explanation and debugging of hazard-triggering events in DNN-based safety-critical systems. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–47.

[28] 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.

[29] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-driven deep learning system testing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 702–713.

[30] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. 2020. Deep learning for 3d point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence* 43, 12 (2020), 4338–4364.

[31] 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.

[32] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon. 2022. An empirical study on data distribution-aware test selection for deep learning enhancement. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 4 (2022), 1–30.

[33] 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.

[34] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. Deepmutation++: A mutation testing framework for deep learning systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1158–1161.

[35] Tianxin Huang and Yong Liu. 2019. 3d point cloud geometry compression on deep learning. In *Proceedings of the 27th ACM international conference on multimedia*. 890–898.

[36] 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.

[37] 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.

[38] Anastasia Ioannidou, Elisavet Chatzilari, Spiros Nikolopoulos, and Ioannis Kompatsiaris. 2017. Deep learning advances in computer vision with 3d data: A survey. *ACM computing surveys (CSUR)* 50, 2 (2017), 1–38.

[39] 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.

[40] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2010), 649–678.

[41] 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).

[42] Ken Kelley and Kristopher J Preacher. 2012. On effect size. *Psychological methods* 17, 2 (2012), 137.

[43] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems* 30 (2017).

[44] 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).

[45] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049.

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

[47] Stephen Kokoska and Daniel Zwillinger. 2000. *CRC standard probability and statistics tables and formulae.* Crc Press.

[48] Martin G Larson. 2008. Analysis of variance. *Circulation* 117, 1 (2008), 115–121.

[49] Wim Lemkens, Prabhjot Kaur, Koen Buys, Peter Slaets, Tinne Tuytelaars, and Joris De Schutter. 2013. Multi RGB-D camera setup for generating large 3D point clouds. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 1092–1099.

[50] 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).

[51] 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.

[52] Xingyu Liu, Mengyuan Yan, and Jeannette Bohg. 2019. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 9246–9255.

[53] Yiling Lou, Junjie Chen, Lingming Zhang, and Dan Hao. 2019. A survey on regression test-case prioritization. In *Advances in Computers*. Vol. 113. Elsevier, 1–46.

[54] Yiling Lou, Dan Hao, and Lu Zhang. 2015. Mutation-based test-case prioritization in software evolution. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 46–57.
[55] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 614–618.
[56] 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.
[57] 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.
[58] 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.
[59] Bilawal Mahmood and SangUk Han. 2019. 3D registration of indoor point clouds for augmented reality. In *ASCE International Conference on Computing in Civil Engineering 2019*. American Society of Civil Engineers Reston, VA, 1–8.
[60] Bilawal Mahmood, SangUk Han, and Dong-Eun Lee. 2020. BIM-based registration and localization of 3D point clouds of indoor scenes using geometric features for augmented reality. *Remote Sensing* 12, 14 (2020), 2302.
[61] Thomas P Minka. 2003. A comparison of numerical optimizers for logistic regression. *Unpublished draft* (2003), 1–18.
[62] 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.
[63] Sebastián Ortega, José Miguel Santana, Jochen Wendel, Agustín Trujillo, and Syed Monjur Murshed. 2021. Generating 3D city models from open LiDAR point clouds: Advancing towards smart city applications. *Open Source Geospatial Science for Urban Studies: The Value of Open Geospatial Data* (2021), 97–116.
[64] 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.
[65] 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.
[66] Mike Papadakis, Christopher Henard, and Yves Le Traon. 2014. Sampling Program Inputs with Mutation Analysis: Going Beyond Combinatorial Interaction Testing. In *Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, March 31 2014-April 4, 2014, Cleveland, Ohio, USA*. IEEE Computer Society, 1–10. https://doi.org/10.1109/ICST.2014.11
[67] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: an analysis and survey. In *Advances in Computers*. Vol. 112. Elsevier, 275–378.
[68] 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).
[69] 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.
[70] François Pomerleau, Francis Colas, Roland Siegwart, et al. 2015. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics* 4, 1 (2015), 1–104.
[71] Michael Prince. 2004. Does active learning work? A review of the research. *Journal of engineering education* 93, 3 (2004), 223–231.
[72] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.
[73] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems* 30 (2017).
[74] Yan Qian, Peng Cao, Wenqing Yin, Fang Dai, Fei Hu, Zhijun Yan, et al. 2017. Calculation method of surface shape feature of rice seed based on point cloud. *Computers and Electronics in Agriculture* 142 (2017), 416–423.
[75] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. Deepmetis: Augmenting a deep learning test set to increase its mutation score. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 355–367.
[76] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 876–888.
[77] Jie Shao, Wuming Zhang, Nicolas Mellado, Pierre Grussenmeyer, Renju Li, Yiming Chen, Peng Wan, Xintong Zhang, and Shangshu Cai. 2019. Automated markerless registration of point clouds from TLS and structured light scanner for heritage documentation. *Journal of Cultural Heritage* 35 (2019), 16–24.

[78] Weijun Shen, Jun Wan, and Zhenyu Chen. 2018. Munn: Mutation analysis of neural networks. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 108–115.
[79] Donghwan Shin, Shin Yoo, Mike Papadakis, and Doo-Hwan Bae. 2019. Empirical evaluation of mutation-based test case prioritization techniques. *Software Testing, Verification and Reliability* 29, 1-2 (2019), e1695.
[80] Martin Simony, Stefan Milzy, Karl Amendey, and Horst-Michael Gross. 2018. Complex-yolo: An euler-region-proposal for real-time 3d object detection on point clouds. In *Proceedings of the European conference on computer vision (ECCV) workshops*. 0–0.
[81] Sandro Sperandei. 2014. Understanding logistic regression analysis. *Biochemia medica* 24, 1 (2014), 12–18.
[82] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
[83] 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.
[84] 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.
[85] 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.
[86] Qian Wang and Min-Koo Kim. 2019. Applications of 3D point cloud data in the construction industry: A fifteen-year review from 2004 to 2018. *Advanced Engineering Informatics* 39 (2019), 306–319.
[87] Yue Wang and Justin M Solomon. 2019. Deep closest point: Learning representations for point cloud registration. In *Proceedings of the IEEE/CVF international conference on computer vision*. 3523–3532.
[88] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (tog)* 38, 5 (2019), 1–12.
[89] 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.
[90] 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.
[91] W Eric Wong, Joseph R Horgan, Saul London, and Aditya P Mathur. 1995. Effect of test set minimization on fault detection effectiveness. In *Proceedings of the 17th international conference on Software engineering*. 41–50.
[92] Wenxuan Wu, Zhongang Qi, and Li Fuxin. 2019. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*. 9621–9630.
[93] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1912–1920.
[94] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120.
[95] 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.
[96] 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.
[97] Jiaying Zhang, Xiaoli Zhao, Zheng Chen, and Zhejun Lu. 2019. A review of deep learning-based semantic segmentation for point cloud. *IEEE access* 7 (2019), 179118–179133.
[98] Zhiyuan Zhang, Yuchao Dai, and Jiadai Sun. 2020. Deep learning based point cloud registration: an overview. *Virtual Reality & Intelligent Hardware* 2, 3 (2020), 222–246.