AudioTest: Prioritizing Audio Test Cases

YINGHUA LI, University of Luxembourg, Luxembourg
XUEQI DANG*, University of Luxembourg, Luxembourg
WENDKÛUNI C. OUÉDRAOGO, University of Luxembourg, Luxembourg
JACQUES KLEIN, University of Luxembourg, Luxembourg
TEGAWENDÉ F. BISSYANDÉ, University of Luxembourg, Luxembourg

Audio classification systems, powered by deep neural networks (DNNs), are integral to various applications that impact daily lives, like voice-activated assistants. Ensuring the accuracy of these systems is crucial since inaccuracies can lead to significant security issues and user mistrust. However, testing audio classifiers presents a significant challenge: the high manual labeling cost for annotating audio test inputs. Test input prioritization has emerged as a promising approach to mitigate this labeling cost issue. It prioritizes potentially misclassified tests, allowing for the early labeling of such critical inputs and making debugging more efficient. However, when applying existing test prioritization methods to audio-type test inputs, there are some limitations: 1) Coverage-based methods are less effective and efficient than confidence-based methods. 2) Confidence-based methods rely only on prediction probability vectors, ignoring the unique characteristics of audio-type data. 3) Mutation-based methods lack designed mutation operations for audio data, making them unsuitable for audio-type test inputs. To overcome these challenges, we propose AudioTest, a novel test prioritization approach specifically designed for audio-type test inputs. The core premise is that tests closer to misclassified samples are more likely to be misclassified. Based on the special characteristics of audio-type data, AudioTest generates four types of features: time-domain features, frequency-domain features, perceptual features, and output features. For each test, AudioTest concatenates its four types of features into a feature vector and applies a carefully designed feature transformation strategy to bring misclassified tests closer in space. AudioTest leverages a trained model to predict the probability of misclassification of each test based on its transformed vectors and ranks all the tests accordingly. We evaluate the performance of AudioTest utilizing 96 subjects, encompassing natural and noisy datasets. We employed two classical metrics, Percentage of Fault Detection (PFD) and Average Percentage of Fault Detected (APFD), for our evaluation. The results demonstrate that AudioTest outperforms all the compared test prioritization approaches in terms of both PFD and APFD. The average improvement of AudioTest compared to the baseline test prioritization methods ranges from 12.63% to 54.58% on natural datasets and from 12.71% to 40.48% on noisy datasets.

CCS Concepts: • **Software and its engineering → Software testing and debugging**; • **Computing methodologies → Neural networks**.

Additional Key Words and Phrases: Test Input Prioritization, Deep Neural Network, Audio Classification, Deep Learning Testing

*Corresponding author.

Authors' Contact Information: Yinghua Li, University of Luxembourg, Luxembourg, Luxembourg, yinghua.li@uni.lu; Xueqi Dang, University of Luxembourg, Luxembourg, Luxembourg, xueqi.dang@uni.lu; Wendkûuni C. Ouédraogo, University of Luxembourg, Luxembourg, Luxembourg, wendkuuni.ouedraogo@uni.lu; Jacques Klein, University of Luxembourg, Luxembourg, Luxembourg, jacques.klein@uni.lu; Tegawendé F. Bissyandé, University of Luxembourg, Luxembourg, Luxembourg, tegawende.bissyande@uni.lu.

ACM 2994-970X/2025/7-ARTISSTA032
https://doi.org/10.1145/3728907

ACM Reference Format:
Yinghua Li, Xueqi Dang, Wendkûuni C. Ouédraogo, Jacques Klein, and Tegawendé F. Bissyandé. 2025. AudioTest: Prioritizing Audio Test Cases. *Proc. ACM Softw. Eng.* 2, ISSTA, Article ISSTA032 (July 2025), 24 pages. https://doi.org/10.1145/3728907

1 Introduction

Audio classification [30] has emerged as a critical research area within the deep learning field, which enables the categorization of audio signals into predefined classes. It plays a crucial role in various applications that impact our daily lives. For example, some modern electric vehicles integrate voice recognition to enable essential functionalities such as unlocking doors or lowering windows in response to users' voice commands [25]. Moreover, some software applications leverage voice recognition for identity verification and access control, such as the WeChat Voiceprint Lock [48].

Ensuring the accuracy and reliability of audio classifiers is crucial because it can directly impact their effectiveness in various fields and thus affect user trust. For example, in the field of mobile devices [37], inaccurate voice recognition can lead to severe consequences. If a smartphone's voice-activated system mistakenly identifies an unauthorized user as an authorized one, it could result in unauthorized access to sensitive personal information. Moreover, an unauthorized user could make unauthorized purchases or transfer funds, severely damaging the owner's financial security.

Testing is typically considered a fundamental practice for ensuring the quality of DNN-based systems [19]. However, there is a significant challenge in testing audio classifiers: labeling audio-type test inputs to verify the correctness of predictions can be costly [75]. This challenge mainly arises from the following factors: 1) manual annotation is still the mainstream method [19]; 2) test sets can be large-scale, increasing labeling efforts; 3) domain-specific knowledge can be required for labelling audio-type test inputs. For example, in the health monitoring field [57], labeling audio data can require medical expertise. When labeling heart murmurs, regular annotators may not be capable of performing these tasks, and it can rely on experts with a medical background for annotation, which further increases the labeling cost.

To address the labelling cost problem, one intuitive solution in the field of DNN testing is to identify and prioritize tests that are more likely to reveal system errors (i.e., inputs that are more likely to be misclassified by the model). This is also known as test prioritization [19, 75, 77]. By identifying such bug-revealing test cases early, testers/developers can label these informative tests sooner to debug and repair the DNN system more efficiently.

In the literature [19, 75, 77], several DNN-oriented test prioritization approaches have been proposed, which can be broadly classified into coverage-based [14, 49, 60, 78], confidence-based [19, 77], and mutation-based approaches [44, 75]. Coverage-based methods, such as CTM [80], adapt traditional software testing techniques to the specific requirements of DNN testing. These methods focus on maximizing the coverage of the DNN's decision logic. Confidence-based approaches leverage the DNN model's prediction confidence for each test input for test prioritization. These methods consider that test samples for which the model's predictions are more uncertain are more likely to be misclassified by the model. For example, DeepGini [19] utilizes Gini Scores for measuring the uncertainty of the model's predictions on tests and leverages this uncertainty to determine the probability of a test being misclassified by the DNN model. Mutation-based approaches utilize mutation analysis to identify potentially misclassified DNN tests. For example, PRIMA [75] and NodeRank [44] proposed new mutation operators, including model and input mutation operators, and utilized mutation testing for test prioritization.

The aforementioned prioritization approaches have the following advantages: 1) Coverage-based approaches aim to maximize coverage metrics, such as neuron activation or path coverage, to

explore diverse behaviors of the DNN model. 2) Confidence-based methods, such as DeepGini, are computationally efficient. 3) Mutation-based approaches have been proven effective in test prioritization. However, when applying the aforementioned approaches to audio-type test cases for test prioritization, they have the following limitations:

- Coverage-based test prioritization methods have been demonstrated to be less effective and efficient compared to confidence-based methods.
- Confidence-based methods prioritize test inputs solely based on the prediction probability vectors of the DNN model without taking into account the unique characteristics of audio-type test data. For example, given a test set T and a model M to be evaluated, confidence-based methods rely solely on the prediction probability vectors generated by M for each $t \in T$. For each test input t, the model M produces a probability vector, $P_M(t) = \{p_1, p_2, \ldots, p_n\}$, where p_i represents the probability that model M predicts t belongs to the i-th class. However, these methods utilize only this probability vector $P_M(t)$ for test prioritization and do not consider the unique characteristics of audio test data when prioritizing tests.
- The mutation-based test prioritization approaches have not designed mutation operations for audio-type data. Therefore, they are not suitable for audio-type test inputs. Specifically, mutation-based approaches propose mutation operators mainly for images, text, or predefined features. For example, shuffling selected characters to generate mutations for text or changing the colors of selected pixels to create mutations for images. However, these mutation operators cannot be directly adapted to audio data, making mutation-based approaches unsuitable for test prioritization on audio data.

In this paper, we propose AudioTest, a novel test prioritization approach specifically designed for audio-type test cases. The core premise of the AudioTest method is that tests closer to misclassified samples are more likely to be misclassified. AudioTest extracts the following four types of features based on the characteristics of audio-type data for test prioritization. The generated four features contribute to solving the limitations of existing test prioritization approaches. Specifically, these features: 1) extract the information of the audio test set itself, overcoming the shortcomings of confidence-based methods, which ignore the dataset information of the test set; 2) enhance effectiveness, addressing the low effectiveness of coverage-based approaches; and 3) are specifically tailored for audio datasets, addressing the limitations of mutation-based methods: these methods are designed specifically for images/text/predefined features and are not suitable for audio data.

- **Time-Domain Features (TD)** TD is extracted from the time-domain representation of audio, including time duration, zero-crossing rate, short-time energy, and amplitude. TD can provide key information about the basic characteristics of the audio signal.
- **Frequency-Domain Features (FD)** FD is extracted from the frequency representation of audio, including spectral centroid, spectral bandwidth, spectral contrast, and spectral flatness. FD can reflect the spectral characteristics of the audio signal.
- **Perceptual Features (PF)** PF reflects the human auditory system's perception of audio, including mel-frequency cepstral coefficients, pitch, and harmonic-to-noise ratio. These features reflect subjective perception.
- **Output Features (OF)** OF captures the model's prediction for an audio test input.

Specifically, AudioTest leverages the four types of features to rank test inputs through the following four steps: **1) Feature Generation** For each test in the test set, AudioTest first generates its four types of features. **2) Feature Concatenation** For each test, AudioTest concatenates all its four types of features to construct a feature vector. **3) Feature Transformation** AudioTest applies a tree-based model to transform the feature vector of each test, which is inspired by the work of He *et al.* [27]. After transformation, misclassified tests cluster more closely together, and correctly

classified tests also cluster more closely. This allows the classification model used by AudioTest to better distinguish misclassified and correctly classified tests, thereby achieving more effective test prioritization. **4) Ranking** AudioTest utilizes a trained model to predict the misclassification probability of each test based on its transformed feature vector. Finally, AudioTest ranks all tests in the test set based on their misclassification probabilities.

AudioTest successfully addresses the limitations of existing test prioritization approaches: 1) To overcome the relatively low effectiveness of coverage-based approaches, AudioTest leverages the unique characteristics of audio data to perform test prioritization, achieving high effectiveness and outperforming all the compared test prioritization approaches. 2) To address the limitation of confidence-based approaches in ignoring the crucial information inherent in the audio test set itself, AudioTest integrates the audio test set information by generating audio-specific features (i.e., time-domain features, frequency-domain features, and perceptual features). AudioTest combines these three dataset-specific features with output features (derived from the model's prediction probability vector) to achieve effective test prioritization. 3) To overcome the limitations of mutation-based methods, where mutation operators are not designed for audio data, AudioTest's feature generation methods are specifically designed for audio data, enabling test prioritization tailored specifically for audio datasets. Compared to the existing test prioritization approaches, AudioTest has the following novelty:

- **Feature generation specifically for audio-type test inputs** AudioTest specifically extracts and leverages features unique to audio-type data, which is not commonly addressed in traditional DNN test prioritization methods.
- **Feature transformation strategy** AudioTest employs a feature transformation method that enhances AudioTest's classification model to better classify misclassified and correctly classified test inputs, thereby improving the effectiveness of test prioritization.
- **Novel and confirmed premise for test prioritization** AudioTest is founded on a novel premise that tests closer to misclassified samples are more likely to be misclassified, which is confirmed in our designed preliminary study (cf. Section 3.1).

AudioTest can be utilized in various audio contexts. For example, in a health monitoring system, DNN models are utilized to predict health conditions based on audio signals such as heartbeats, breathing, and coughs. Incorrect predictions can lead to significant issues. For instance, if the system mistakenly identifies an abnormal heartbeat as normal, it can fail to alert healthcare providers, resulting in undiagnosed health conditions and delayed treatment. In this context, AudioTest can be used to identify and prioritize audio samples that are more likely to be misclassified by the model. These potentially misclassified samples can be prioritized for manual checks, reducing the instances of delayed treatment and allowing patients to receive more timely care.

We evaluate AudioTest's performance utilizing 96 subjects (i.e., paired audio-type datasets and DNN models). The evaluation includes both natural and noisy datasets. The selected datasets and models for evaluation are widely used in the field of audio classifiers [8, 20, 83]. We utilized two classical evaluation metrics, Percentage of Fault Detection (PFD) and Average Percentage of Fault Detected (APFD), to evaluate AudioTest. These metrics are widely adopted to assess the effectiveness of test prioritization [19, 77]. The experimental results demonstrate that AudioTest outperforms all the compared test prioritization approaches in terms of both PFD and APFD. The average improvement of AudioTest over the baseline prioritization methods ranges from 12.63% to 54.58% on natural datasets and from 12.71% to 40.48% on noisy datasets. We publish our dataset, results, and tools to the community on Zenodo[1]. To sum up, our work has the following major contributions:

[1]https://zenodo.org/records/13961855

- **Approach.** We propose AudioTest, a novel test prioritization approach specifically designed for audio-type test cases.
- **Study.** We evaluate AudioTest utilizing 96 subjects involving natural and noisy test inputs. We compare AudioTest with several existing DNN test prioritization approaches. Our experimental results demonstrate the effectiveness of AudioTest.
- **Performance Analysis.** We evaluate the contributions of different types of features to AudioTest's effectiveness through a carefully designed ablation study. Additionally, we explore to what extent the variation of parameter settings impacts AudioTest's effectiveness to evaluate its stability.

The rest of this paper is organized as follows. Section 2 introduces the background on deep learning for audio classification and test prioritization techniques. Section 3 presents the specific methodology of AudioTest. Section 4 exhibits the design of our study. Section 5 presents the experimental results and analysis. Section 6 discusses the potential threats to the validity of AudioTest, and Section 7 reviews the related work. Finally, Section 8 concludes the paper.

2 Background

2.1 Deep Learning for Audio Classification

Audio classification [30] analyzes audio signals, assigning them to categories based on features. In the literature, Kong *et al.* [42] proposed PANN, a pre-trained audio neural network trained on the large-scale AudioSet dataset [23]. PANN employs a variety of convolutional neural network (CNN) architectures, which combine raw waveform and log-mel spectrogram inputs to capture rich time-frequency representations. These models efficiently extract hierarchical audio features from input spectrograms. Desplanques *et al.*[12] proposed ECAPA-TDNN, an advanced speaker verification model that builds upon the x-vector framework [67]. The model processes audio data by extracting Mel-Frequency Cepstral Coefficients as input features. The generated embeddings are fed into fully connected layers that map the representations into class probabilities. Martinez *et al.* [52] developed a speech intelligibility model that processes audio by leveraging automatic speech recognition (ASR) techniques. The model extracts phoneme probabilities using a Time Delay Neural Network architecture, which is specifically designed to capture temporal dependencies in speech signals. Wang *et al.* [74] proposed CAM++, an efficient speaker verification network for audio classification. CAM++ processes input spectrograms using a front-end convolution module to extract time-frequency features and pass them through a densely connected TDNN backbone to capture temporal patterns. Chen *et al.* [8] proposed ERes2Net, a speaker verification framework designed to leverage both local and global feature fusion. By augmenting the original Res2Net architecture, ERes2Net integrates an Attentional Feature Fusion (AFF) mechanism to dynamically emphasize significant features.

2.2 Test Input Prioritization for DNNs

Test prioritization is crucial in software testing [4, 45, 58, 71, 72, 80, 81] for determining the optimal order of unlabelled tests. In DNN testing [9, 19, 75], test prioritization strategies fall into three main categories: coverage-based [80], confidence-based [19, 77], and mutation-based [75] approaches. Coverage-based methods, such as CTM [80], focus on maximizing the coverage of the DNN's decision logic. Confidence-based methods prioritize tests based on the DNN's uncertainty, with approaches such as DeepGini [19], Vanilla Softmax, PCS, and Entropy [77] showing high effectiveness. Mutation-based approaches, such as PRIMA [75], use mutation analysis for test prioritization. In addition to the aforementioned three types of test prioritization methods, there are also methods that leverage a combination of these categories. For example, Wei *et al.* proposed

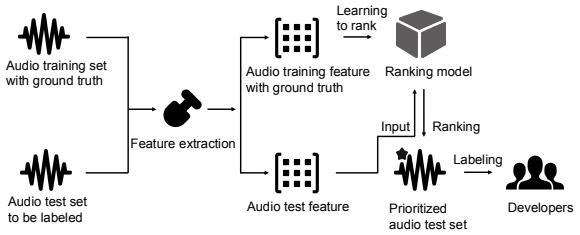


Fig. 1. Workflow of testing audio systems

EFFIMAP [76], an efficient test prioritization method that integrates both the coverage-based principle and mutation analysis. First, EFFIMAP generates model mutants, and leverages execution traces to predict whether a test case can kill a mutant. Then, the EFFIMAP method evaluates the effectiveness of test cases comprehensively through a metric called killing coverage, which refers to the proportion of mutants killed by a test case. EFFIMAP prioritizes tests based on this killing coverage. However, when applying the aforementioned approaches to audio-type tests, several limitations have emerged. Coverage-based methods have been shown to perform less efficiently and effectively compared to confidence-based approaches. Confidence-based methods rely solely on the prediction probability vectors output by the DNN model to perform test prioritization. These approaches do not take into account the crucial information of the audio test set itself for test prioritization. Furthermore, mutation-based test prioritization approaches propose new mutation operators for test prioritization, and these operators are mostly designed for images, text, or predefined features. Such operators cannot be directly applied to audio data. As a result, mutation-based approaches are not suitable for test prioritization in audio datasets. In this paper, we propose AudioTest, a test prioritization approach specifically designed for audio-type tests. Figure 1 illustrates its fundamental workflow for testing audio classification systems. First, AudioTest extracts features from the input training set. Based on the extracted features, AudioTest trains a ranking model that predicts the probability of a test being misclassified based on its feature representation. Once the ranking model is trained, it evaluates a given test set by calculating the misclassification score for each test. This enables AudioTest to rank all tests. Developers can then focus more on labeling the potentially misclassified tests.

3 Approach

3.1 Preliminary Study

The key premise of AudioTest is that *tests closer to misclassified samples are more likely to be misclassified*. To the best of our knowledge, we are the first to propose this premise, and no prior work has proposed this premise or proposed an assumption requiring validation. We conducted a carefully designed preliminary study to validate the proposed premise.

Experimental Design We first calculated the number of misclassified tests among the nearest n tests surrounding each misclassified test and took the average. Then, we calculated the number of misclassified tests surrounding each correctly classified test and took the average. We visualized the experimental results (cf. Figure 2) to support the premise. Specifically, Formula 1 calculates the average number of misclassified tests among the nearest n tests ($n = 30, 40, 50, \ldots, 150$) surrounding each misclassified sample. For example, if $n = 100$, it means that we calculated the average number of misclassified tests among 100 tests surrounding a misclassified sample.

$$\bar{M} = \frac{1}{N} \sum_{i=1}^{N} M_i \tag{1}$$

where N refers to the total number of misclassified samples. M_i represents the number of misclassified samples around the i-th misclassified sample.

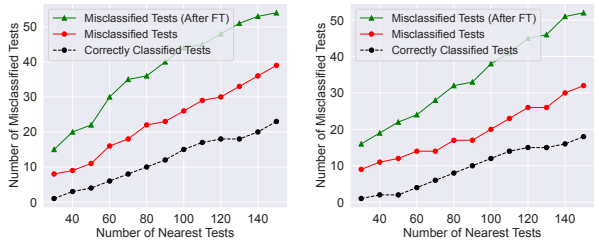
Moreover, Formula 2 calculates the average number of misclassified tests among the nearest n tests surrounding each correctly classified sample.

$$\bar{C} = \frac{1}{K} \sum_{i=1}^{K} M_i \tag{2}$$

where K refers to the total number of correctly classified samples. M_i represents the number of misclassified samples around the i-th correctly classified sample.

Our preliminary study was conducted using datasets that are commonly employed in the field of audio classification (cf. Section 4.2.1). These datasets were extensively used in academic research [21, 22, 26, 28, 47, 65]. Moreover, these datasets have attracted significant attention from researchers and engineers for in-depth analysis on the widely recognized data science competition platform Kaggle [38]. Moreover, to ensure the quality of the collected data, we conducted a manual review of the integrity and quality of each dataset, ensuring the links were correct and the content was accurate. Additionally, we verified the actual usage of these datasets within the research community, confirming that they have been extensively utilized by researchers. Through this, we aim to guarantee the accuracy of the data as well as ensure its reliability and broad recognition in both academic and practical applications.

Results and findings Figure 2 shows the results of the preliminary study. The X-axis represents selecting n tests closest to the original test. The Y-axis represents the number of misclassified tests among the n tests closest to the original test. The red curve represents the average number of misclassified tests around a misclassified sample when selecting the n tests closest to the original test. The black curve represents the average number of misclassified tests around a correctly classified sample. From Figure 2, we see that the red curve (representing the number of misclassified tests around a misclassified sample) consistently lies above the black curve (representing the number of misclassified tests around a correctly classified sample), and this trend remains consistent across different neighborhood sizes N. This indicates that the density of misclassified tests around misclassified samples is higher than that around correctly classified samples, suggesting that misclassified tests are more likely to cluster. This observation forms the basis of AudioTest's premise: tests closer to misclassified samples are more likely to be misclassified.



(a) UrbanSound with CAMPPlus (b) UrbanSound with EcapaTdnn

Fig. 2. Average number of misclassified tests around a misclassified/correctly classified sample

3.2 Overview

Figure 3 shows AudioTest's workflow. It takes an audio test set T and a model M as input, and outputs a sorted test set T' with samples more likely to be misclassified prioritized at the top. Details of each step are presented in Sections 3.3 to 3.6.

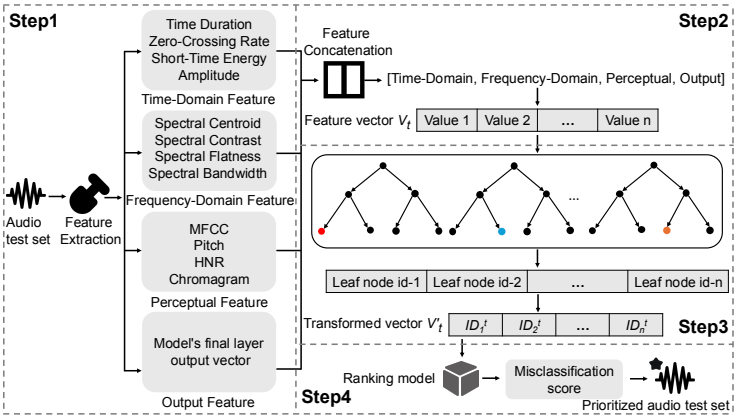


Fig. 3. Overview of AudioTest

❶ **Step1: Feature Extraction** For each test sample t in the test set T, AudioTest utilizes the tool Librosa [53] (a Python library for audio analysis) to process the audio data and extract features.

❷ **Step2: Feature Concatenation** Next, AudioTest concatenates the extracted features for each test t. After concatenation, each test t has a corresponding feature vector, denoted as V_t, in the format [value 1, value 2, ..., value n].

❸ **Step3: Feature Transformation** For each test t's feature vector V_t, AudioTest performs feature transformation. Specifically, AudioTest uses N decision trees to transform V_t to a new vector V_t'. The objective is to enhance the classification model used by AudioTest in its ability to classify misclassified samples and correctly classified samples, thereby enhancing the effectiveness of test prioritization.

❹ **Step4: Ranking** We trained a classification model. By inputting the transformed feature vector of each test $t \in T$ into the classification model, we extracted an intermediate result (a probability value). This probability value indicates the probability of a test being misclassified by the model. AudioTest ranked all the tests based on this probability value and output the sorted test set T'.

The ranking process in the context of audio classification contributes to two main aspects: improving the efficiency of model debugging and enhancing the audio classification model. Specifically, 1) After ranking, tests with a higher probability of being misclassified are prioritized higher. These tests are more likely to reveal potential faults in the audio classification model. By prioritizing such tests, developers and testers can focus more on these fault-revealing tests and ensure early diagnosis, thereby improving debugging efficiency. 2) These fault-revealing tests can be used to retrain the audio classification model to improve its performance. Existing studies have proven that such a retraining strategy is effective in enhancing the DNN models [19].

3.3 Step1: Feature Extraction

In the first step, for each test $t \in T$, AudioTest employs Librosa [53], a widely-used Python library for audio analysis, to extract features from the test (an audio). These features provide a detailed representation of the audio signals. For each test $t \in T$, we extracted four different types of features:

- **Time-Domain Features (TD)** Time-domain features are extracted directly from the time-domain representation of an audio-type test input, which can include time duration, zero-crossing rate, short-time energy, and amplitude. Time duration is the total length of the audio. The zero-crossing rate reflects the number of times the audio crosses the zero axis, also referred to as the frequency of the signal. Short-time energy is the energy of the audio signal calculated over a short time

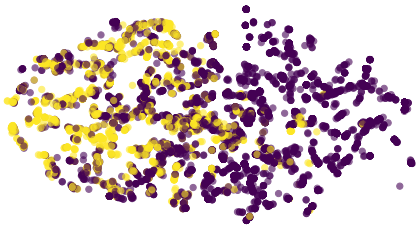


Fig. 4. Spatial distribution of transformed feature vectors: misclassified tests (yellow points) vs. correctly classified tests (purple points)

window. Amplitude represents the strength of the signal. Time-domain features can provide key information about the basic characteristics of the audio signal.

- **Frequency-Domain Features (FD)** Frequency-domain features are extracted from the frequency representation of the audio signal. These features provide important information about the signal's frequency content and distribution. FD include spectral centroid, spectral bandwidth, spectral contrast, and spectral flatness, which reflect the center position of the spectrum, the width of the distribution, the contrast between different frequency bands, and the flatness of the spectrum, respectively. These features can reflect the spectral characteristics of the audio signal.
- **Perceptual Features (PF)** Perceptual features reflect the subjective perception of audio by the human auditory system. They include four crucial aspects: 1) Mel-Frequency Cepstral Coefficients, reflecting human sound perception; 2) Chromagram, representing the energy distribution of the twelve semitones for analysis; 3) Pitch, reflecting the fundamental frequency; and 4) Harmonic-to-Noise Ratio, measuring the ratio of harmonic to noise components in the signal. Perceptual features can reflect the subjective perception of sound by the human auditory system.
- **Output Features (OF)** Output features capture the model's prediction information for an audio-type test input. To obtain the output features of a test, we input it into the target model, and the model will output a probability vector representing the probabilities of the test belonging to each class. Here, "each class" refers to the predefined labels that the model is trained to classify a test input into. For example, in the audio dataset UrbanSound [65] used in our study, class examples include: "air conditioner", "car horn", and "children playing".

3.4 Step2: Feature Concatenation

For each test $t \in T$, AudioTest concatenates all four types of features to generate a feature vector, denoted as V_t. The format of V_t is: [TD, FD, PF, OF], which refers to the four types of features respectively. The size of V_t ranges from 191 to 231 dimensions. Specifically, the TD, FD, and PF features have sizes of 4, 50, and 135 dimensions, respectively, while the OF features range from 2 to 41 dimensions (The dimensions of OF vary because the output dimensions differ across different datasets). We added a specific example to illustrate the extracted features: TD: $[t_1, t_2, \ldots, t_i]$; FD: $[f_1, f_2, \ldots, f_j]$; PF: $[p_1, p_2, \ldots, p_k]$; OF: $[o_1, o_2, \ldots, o_m]$. After feature concatenation, the combined vector is:

$$[t_1, t_2, \ldots, t_i, f_1, f_2, \ldots, f_j, p_1, p_2, \ldots, p_k, o_1, o_2, \ldots, o_m]$$

where each element corresponds to a specific value. Here, i, j, k, and m represent the dimensions of TD, FD, PF, and OF, respectively.

3.5 Step3: Feature Transformation

Objective. In the third step, for each test input $t \in T$, we transform its original feature vector V_t into V_t' using a feature transformation strategy. The objective of this transformation is to make the

transformed features more useful for implementing AudioTest's premise for test prioritization: tests closer to misclassified samples are more likely to be misclassified. Specifically, In Section 3.1, we validated this premise. As shown in Figure 2, the red curve represents the average number of misclassified tests around a misclassified sample, while the black curve represents the average number of misclassified tests around correctly classified samples. The gap between the two curves confirms that misclassified samples tend to cluster together, thereby validating the premise.

Approach. Our feature transformation is based on the decision tree algorithm [64].

Decision tree algorithm. Decision trees are a fundamental ML approach commonly used for classification tasks. The model constructs a tree structure that recursively divides the dataset based on feature values. Each node in the tree represents a decision rule, while the branches indicate possible outcomes of these decisions. Once a decision tree is trained, when a sample is given, the prediction process begins at the root node. The model compares the sample's feature values with the decision rules at each node and continues down the tree until reaching a leaf node, where the predicted value is generated. We adopted the decision tree algorithm for feature transformation because it helps bring misclassified tests closer in the feature space. This is achieved by the tree model grouping similar samples into adjacent leaf nodes. Consequently, misclassified tests are clustered together in the feature space, making them spatially closer. Through this approach, the classification model employed by AudioTest can more effectively differentiate between misclassified and correctly classified tests, thereby improving the effectiveness of test prioritization.

Feature transformation workflow. Given a test input t, we obtained its feature vector in the previous step. To perform feature transformation on this feature vector, we first constructed N trees, each containing three types of nodes: root nodes, intermediate nodes, and leaf nodes. The root and intermediate nodes contain two pieces of information: the vector index used for decision-making and the threshold value. The leaf nodes contain one piece of information, which is the leaf node ID (numeric value). Suppose we place the test vector t into the first tree of the N decision trees. Given the feature vector of t, denoted as $V_t = \{v_0, v_1, \ldots, v_n\}$, the tree makes decisions starting from the root node. If the element of V_t that corresponds to the root node is less than the root node's threshold, the vector falls into the left intermediate node; otherwise, it falls into the right intermediate node. It is important to note that which element of V_t corresponds to which node in the tree, as well as the value of the thresholds, is determined by the decision tree itself during the learning process. After reaching the first intermediate node, the decision tree checks whether the value of the element of V_t that corresponds to the first intermediate node is less than the node's threshold. If it is, the vector falls into the left node; otherwise, it falls into the right node. This decision-making process continues until the test input t reaches a leaf node. We then record the leaf node ID as ID_1, where the 1 represents the first tree. Since we have constructed N trees, the test input t can obtain N leaf node IDs. Therefore, the new feature vector for t is recorded as $\{ID_1, ID_2, \ldots, ID_N\}$, where ID_n represents the leaf node ID in which test t falls in the n-th tree.

Underlying principle. Utilizing the aforementioned feature transformation approach for test prioritization is inspired by the work of He *et al.* [27], which demonstrated that feature transformation can effectively improve the performance of classification algorithms. Specifically, our feature transformation strategy can enhance the classification model used by AudioTest in improving its ability to classify misclassified and correctly classified samples. After feature transformation, tests that are more likely to be misclassified are clustered closer to each other, thereby also facilitating our premise for test prioritization, which is that *tests closer to misclassified samples are more likely to be misclassified*. Figure 4 shows the spatial distribution of the tests after feature transformation (visualized using t-SNE [73]). Each point represents a test. Yellow points indicate misclassified tests, while purple points indicate correctly classified tests. In Figure 4, the yellow points tend to cluster towards the left side of the figure, and the purple points tend to cluster towards the right side. There

is a tendency for yellow points to be surrounded by more yellow points and purple points to be surrounded by more purple points. This intuitively demonstrates that after feature transformation, tests misclassified by the model are more likely to be surrounded by other misclassified tests.

The reason why feature transformation works lies in that it makes misclassified tests more clustered. Specifically, feature transformation allows similar samples (misclassified tests) to fall into adjacent leaf nodes on the decision tree, making them more clustered in the space. This could contribute to AudioTest's ability to better leverage the premise (i.e., tests close to misclassified samples are more likely to be misclassified) for test prioritization. To provide a more intuitive illustration of the effects of feature transformation, in Figure 2, we present, before and after feature transformation, the average number of misclassified tests around each misclassified test. Specifically, in Figure 2, the X-axis represents the n tests that are closest to the original test, while the Y-axis represents the number of misclassified tests among these n tests. The red curve indicates, among the n closest tests to a misclassified sample, the average number of misclassified tests. The green curve represents, after feature transformation, the average number of misclassified tests. In Figure 2, we see that the green line consistently stays above the red line. This exhibits that after feature transformation, the number of misclassified tests surrounding a misclassified sample increases. In other words, the feature transformation process makes the misclassified samples more clustered, which could contribute to AudioTest's ability to better leverage the premise (i.e., tests close to misclassified samples are more likely to be misclassified) to perform test prioritization.

3.6 Step4: Ranking

Building on the three steps (i.e., feature extraction, feature concatenation, and feature transformation) described above, AudioTest generates a final vector V'_{t_i} for each test $t_i \in T$. Subsequently, a ranking model was trained to predict the misclassification probability of each test t_i based on its vector V'_{t_i}. The process of training and constructing the ranking model is detailed below.

Model Training Given an audio classification model M and a dataset A, the dataset is initially divided into two partitions: a training set R and a test set T, following a 7:3 ratio [56]. The test set T is reserved for evaluating the performance of AudioTest and remains unchanged throughout the training process. From the training set R, we construct a new training set R': 1) we performed feature extraction, feature concatenation, and feature transformation (cf. Sections 3.3 to 3.5) on each sample in the original training set R, generating new feature vectors for each sample. These feature vectors form the training features of the new training set R'. 2) Each sample in R was labeled based on the following rule: misclassified tests were labeled as 1, and correctly classified tests were labeled as 0. These labels serve as the label set for the new training set R'. Finally, the ranking model was trained using the newly constructed training set R'.

Model adjustment The original ranking model is a binary classification model. Once the ranking model is trained, we make adjustments to it so that it outputs the misclassification probability (an intermediate value) of a given test instead of the test's label. This probability value is utilized for ranking. A higher probability value indicates that the test is more likely to be misclassified.

Evaluation Metrics We evaluated the accuracy of the ranking model across three datasets. On the Actor dataset, the accuracy ranges from 71.9% to 81.4%. For the FSD dataset, it ranges from 79.7% to 87.3%, while for the UrbanSound dataset, the accuracy falls between 85.4% and 91.5%.

4 Study Design

4.1 Research Questions

In the following, we present the research questions of this study. They are organized into the following four aspects:

Table 1. Audio classification datasets and models

ID	Dataset	Model	Type
1	Actor	CAMPPlus	Natural, Brown, Gaussian, White
2	Actor	EcapaTdnn	Natural, Brown, Gaussian, White
3	Actor	ERes2Net	Natural, Brown, Gaussian, White
4	Actor	PANNS	Natural, Brown, Gaussian, White
5	Actor	ResNetSE	Natural, Brown, Gaussian, White
6	Actor	TDNN	Natural, Brown, Gaussian, White
7	UrbanSound	CAMPPlus	Natural, Brown, Gaussian, White
8	UrbanSound	EcapaTdnn	Natural, Brown, Gaussian, White
9	UrbanSound	ERes2Net	Natural, Brown, Gaussian, White
10	UrbanSound	PANNS	Natural, Brown, Gaussian, White
11	UrbanSound	ResNetSE	Natural, Brown, Gaussian, White
12	UrbanSound	TDNN	Natural, Brown, Gaussian, White
13	FSD	CAMPPlus	Natural, Brown, Gaussian, White,Impulse,Sawtooth, Sine, Triangle
14	FSD	EcapaTdnn	Natural, Brown, Gaussian, White,Impulse,Sawtooth, Sine, Triangle
15	FSD	ERes2Net	Natural, Brown, Gaussian, White,Impulse,Sawtooth, Sine, Triangle
16	FSD	PANNS	Natural, Brown, Gaussian, White,Impulse,Sawtooth, Sine, Triangle
17	FSD	ResNetSE	Natural, Brown, Gaussian, White,Impulse,Sawtooth, Sine, Triangle
18	FSD	TDNN	Natural, Brown, Gaussian, White,Impulse,Sawtooth, Sine, Triangle

- **RQ1: How effective and efficient is AudioTest in prioritizing audio-type test inputs?**
  We compared the effectiveness of AudioTest with other test prioritization methods. We evaluated the performance of AudioTest from three perspectives. **1) Effectiveness.** We evaluated AudioTest using 18 audio-type subjects, detailed in Table 1. **2) Statistical analysis.** Due to the randomness in model training, we conducted a statistical analysis by repeating all experiments 30 times [2] and reporting the average results. We calculated the p-value and effect size to demonstrate the statistical significance of our experimental results. First, we utilized the Mann-Whitney U test [54] to compute the p-value of the repeated experimental results. Second, we use Cliff's Delta d [31] to measure the effect size.
- **RQ2: How effective is AudioTest in prioritizing noisy audio tests?**
  We compared the effectiveness of AudioTest and other test prioritization methods on noisy datasets generated by seven different noise generation techniques. We utilized seven noise generation techniques (Section 4.3) to generate noisy test sets. On these noisy test sets, we evaluated AudioTest and other test prioritization methods.
- **RQ3: Do each type of features contribute to the effectiveness of AudioTest?**
  In AudioTest, for an audio-type test, we generated four types of features for test prioritization. In this research question, we conducted an ablation study to demonstrate that each type of features contributes to AudioTest. The original AudioTest has four types of features for test prioritization. In the ablation study, we removed one feature at a time and observed whether the effectiveness decreased.
- **RQ4: How do main parameters in AudioTest impact its effectiveness?**
  We investigate the influence of the main parameters (i.e., *dimension*, *max_depth*, and *colsample_bynode*) on the effectiveness of AudioTest to verify whether its effectiveness can remain stable when the main parameters change. We changed the value of these parameters and observed the impact on the effectiveness of AudioTest.

4.2 Models and Datasets

We evaluated AudioTest on different types of audio-type test inputs, including natural audio tests and noisy audio tests. In total, we built 96 subjects.

4.2.1 Audio Datasets. We utilized three datasets for evaluation. We adopted these audio datasets because they are widely used in the field of audio classification. On one hand, these datasets have been extensively utilized in academia in recent years [21, 22, 26, 28, 47, 65]. On the other hand, on the renowned data science competition platform Kaggle [38], these datasets have garnered significant attention from researchers and engineers for in-depth studies.

- **UrbanSound** [65] The UrbanSound dataset is utilized for urban sound classification. It includes 8,732 sound excerpts categorized into 10 classes, such as air conditioner and children playing.
- **FSD** [21] FSD is an audio dataset containing 11,073 audio files classified into 41 classes from the AudioSet Ontology. Examples of the labels are: Applause, Electric piano, Fireworks, and Flute.
- **Actor** [47] The Actor dataset is used for audio emotion classification tasks. It contains 1,440 files from 24 actors. The dataset classifies emotions into different types and intensities.

4.2.2 DNN Models. We utilize six audio classification model to evaluate AudioTest: CAMP-Plus [74], EcpaTdnn [12], ERes2Net [8], PANNs [42], ResNetSE [32], and TDNN [52]. A detailed description of these models can be found in Section 2.1. We selected these DNN models to evaluate AudioTest because they have been widely utilized in related literature [8, 11, 74, 83]. Moreover, these models demonstrate diversity and support various types of audio classification scenarios, which could contribute to validating the generalizability of AudioTest across different scenarios.

4.3 Noise Generation Techniques

We used Pydub [35] to add seven types of noise to the original datasets to construct noisy datasets.

- **White Noise** White noise is a type of noise that has equal power across all frequencies.
- **Brown Noise** Brown noise is a type of noise whose power spectral density is inversely proportional to the square of the frequency. Brown noise is commonly used to simulate natural sounds such as ocean waves and wind.
- **Impulse Noise** Impulse noise consists of short bursts of high-intensity signals. It is mainly used to simulate situations such as electromagnetic interference or mechanical impacts.
- **Sine Wave Synthesized Noise** Sine wave synthesized noise combines multiple sine waves of different frequencies and amplitudes. It is mainly used to simulate complex acoustic environments for applications such as audio testing.
- **Triangle Wave Synthesized Noise** Triangle Wave Synthesized Noise is generated using a triangle wave. A triangular wave is a periodic waveform similar to an isosceles triangle, which is characterized by linear rise and fall.
- **Sawtooth Wave Synthesized Noise** Sawtooth wave synthesized noise is generated using a sawtooth waveform. It is typically used to create aggressive sound effects.
- **Gaussian Noise** Gaussian noise is a common type of random noise, with its amplitude distribution following a Gaussian distribution. Gaussian noise can effectively represent various random noises found in nature.

4.4 Measurements

We employed two classical metrics, Average Percentage of Fault Detection (APFD) and Percentage of Fault Detected (PFD), for evaluation. These metrics were selected due to their widespread recognition in the field of test prioritization [19, 77, 80]. Influential studies, such as DeepGini (ISSTA 2020) [19] and a replicability study on test prioritization (ISSTA 2022) [77], have demonstrated their relevance and effectiveness as benchmarks for evaluating test prioritization techniques.

(1) APFD is a widely-adopted metric for assessing the effectiveness of test prioritization methods (cf. Formula 3). A higher APFD value indicates faster detection of misclassifications.

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{k} o_i}{kn} + \frac{1}{2n} \tag{3}$$

where n denotes the number of tests in the test set, k represents the number of misclassified inputs, and o_i is the index of the i-th misclassified test within the prioritized test set. Following prior work [19], we normalize the APFD values to the range [0, 1].

(2) PFD calculates the proportion of detected misclassified tests among all misclassified tests, as presented in Formula 4. In our experiments, we calculated the PFD values when prioritizing the top 10%, 20%, 30%, 40%, 50%, and 60% of test inputs, respectively.

$$PFD = \frac{\#M_{detect}}{\#M} \tag{4}$$

where $\#M_{detect}$ refers to the number of detected misclassified test inputs. $\#M$ denotes the total number of misclassified tests in the test set.

4.5 Compared Approaches

We compare AudioTest with five test prioritization approaches collected from the literature [19, 77]: DeepGini [19], VanillaSM [77], Prediction-Confidence Score [77], Entropy [77], and Random Selection [17]. We selected these compared approaches because: 1) these methods can all be applied to test prioritization in audio classification scenarios, and 2) the performance of these methods has been validated in existing studies [19, 77].

- **DeepGini** [19] DeepGini prioritizes tests based on the model's prediction uncertainty for the tests (cf. Formula 5). Tests with higher uncertainty are considered more likely to be misclassified.

$$G(t) = 1 - \sum_{i=1}^{N} (p_i(t))^2 \tag{5}$$

  where $p_i(t)$ denotes the probability that the model predicts the test t belongs to class i. N denotes the total number of classes predicted by the model.
- **Prediction-Confidence Score (PCS)** [77] PCS computes the model's prediction confidence for each test by calculating the difference between the probabilities of the model's top two predictions for a given input t. A smaller PCS value indicates a higher likelihood of misclassification.
- **VanillaSM** [77] VanillaSM measures the difference between the model's highest predicted probability for a test and 1. A larger VanillaSM value indicates a higher misclassification probability.
- **Entropy** [77] Entropy measures the entropy of the model's softmax likelihood for a test. A test with high entropy values is considered more likely to be misclassified.
- **Random selection** [17] Utilizing random selection, the execution order of test inputs is randomized.

4.6 Implementation and Configuration

We implemented AudioTest in Python 3.8.19, utilizing the PyTorch 1.13.1 framework [59] and XGBoost 2.1.0 [7] for the ranking model. In the feature transformation process, we used 100 trees to enhance the original feature vector of each test (the size of the transformed vector for each test is 100). We conduct the experiments on a 2.6 GHz Intel Xeon Gold 6132 CPU with an NVIDIA Tesla V100 16G SXM2 GPU. For data analysis, we utilized a MacBook Pro laptop running macOS Sonoma 14.3, equipped with an Intel Core i9 CPU and 64 GB of RAM. Additionally, we integrated existing implementations of the compared methods [19, 77] into our experimental pipeline.

5 Experimental Results and Analysis

5.1 RQ1: Effectiveness and Efficiency of AudioTest on Natural Test Inputs

Objectives: We evaluate the effectiveness and efficiency of AudioTest, comparing it with various existing test prioritization approaches. Our investigation is conducted through two sub-questions:

- **RQ-1.1** How effective is AudioTest in prioritizing audio test inputs?
- **RQ-1.2** How efficient is AudioTest?

Table 2. Effectiveness comparison on natural data (APFD) with best results highlighted in gray

Data	Model	Approach					
		Random	DeepGini	Entropy	PCS	VanillaSM	AudioTest
Actor	CAMPPlus	0.536	0.585	0.585	0.585	0.585	0.734
	EcapaTdnn	0.496	0.645	0.645	0.645	0.645	0.734
	ERes2Net	0.506	0.629	0.629	0.629	0.629	0.729
	PANNS	0.507	0.589	0.589	0.589	0.589	0.739
	ResNetSE	0.519	0.628	0.628	0.628	0.628	0.707
	TDNN	0.522	0.626	0.626	0.626	0.626	0.729
FSD	CAMPPlus	0.488	0.722	0.711	0.727	0.726	0.786
	EcapaTdnn	0.507	0.746	0.737	0.741	0.746	0.797
	ERes2Net	0.493	0.672	0.665	0.673	0.674	0.739
	PANNS	0.507	0.720	0.714	0.716	0.722	0.765
	ResNetSE	0.503	0.705	0.696	0.707	0.709	0.764
	TDNN	0.498	0.699	0.691	0.698	0.701	0.742
UrbanSound	CAMPPlus	0.498	0.736	0.735	0.729	0.734	0.829
	EcapaTdnn	0.486	0.734	0.731	0.736	0.737	0.822
	ERes2Net	0.466	0.732	0.732	0.725	0.731	0.842
	PANNS	0.510	0.744	0.737	0.746	0.748	0.838
	ResNetSE	0.496	0.710	0.707	0.706	0.710	0.841
	TDNN	0.509	0.758	0.755	0.756	0.759	0.845

Table 3. Overall comparison results across all natural subjects (APFD)

Approach	# Best cases	Average APFD	Improvement (%)
Random	0	0.502	54.58
DeepGini	0	0.687	12.95
Entropy	0	0.684	13.45
PCS	0	0.686	13.12
VanillaSM	0	0.689	12.63
AudioTest	18	0.776	-

Table 4. Statistical analysis on natural datasets

	Random	DeepGini	Entropy	PCS	VanillaSM
p-value	3.12×10^{-8}	1.21×10^{-6}	6.33×10^{-7}	4.87×10^{-7}	1.46×10^{-7}
effect size	1.0	0.877	0.901	0.911	0.952

Table 5. Effectiveness comparison on natural tests (PFD)

Approach	10%	20%	30%	40%	50%	60%
Random	0.104	0.201	0.302	0.401	0.502	0.605
DeepGini	0.207	0.383	0.548	0.679	0.785	0.866
Entropy	0.205	0.376	0.540	0.669	0.777	0.863
PCS	0.201	0.379	0.549	0.678	0.783	0.865
VanillaSM	0.207	0.385	0.551	0.681	0.784	0.865
AudioTest	0.301	0.548	0.728	0.839	0.905	0.942

Results: We present the experimental results of RQ1.1 as follows:

Effectiveness of AudioTest Table 2 compares AudioTest with other test prioritization methods using the APFD metric across natural subjects. The best-performing approach for each case is highlighted in grey. We see that AudioTest performs the best across all cases. Table 3 further shows the average APFD value for AudioTest and its relative improvement compared to the comparative methods. We see that AudioTest achieves an average APFD of 0.776, with an average improvement of 12.63%~54.58% over the comparative methods. Table 5 compared AudioTest with existing approaches using the PFD metric. In Table 5, the columns labeled "10%, 20%, 30%, ..." represent the proportion of tests executed. For example, "10%" shows the PFD values of each test prioritization approach when the top 10% of the tests are executed. In other words, it reflects the ratio of misclassified tests detected by each prioritization method after executing 10% of the entire test set. We see that, across different proportions of prioritized tests, AudioTest consistently outperforms all the compared methods. Additionally, we see that in the binary classification dataset Actor, all uncertainty-based methods have the same APFD results. We explain the reasons below. First, in binary classification, when a model predicts $(0.5, 0.5)$ for a test input, it indicates that the prediction is most uncertain [19]. Therefore, the uncertainty of a test t with prediction $(p, 1-p)$ depends solely on p, with values closer to 0.5 considered more uncertain. Consequently, regardless

Table 6. Time cost of AudioTest and the compared test prioritization approaches

Time cost	Approach							
	AudioTest			Random	DeepGini	Entropy	PCS	VanillaSM
Feature generation	4.9 min(8 threads)	2.4 min(16 threads)	1.2 min(32 threads)	-	-	-	-	-
Ranking model training	16 s	16 s	16 s	-	-	-	-	-
Prediction	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s

of the uncertainty-based method used, the final ranking depends only on the value of p for each test. As a result, all uncertainty-based approaches produce the same ranking outcomes for a given test set, leading to identical APFD.

Statistical Analysis The results of the statistical analysis are presented in Table 4. We see that all the p-values between AudioTest and the compared approaches are less than 10^{-5}. According to prior work [51], which considers a p-value less than 10^{-5} to indicate a statistically significant difference between two data sets, we conclude that the improvement of AudioTest compared to existing approaches is statistically significant. Moreover, the effect size between AudioTest and each test prioritization approach is greater than 0.33. According to Cliff's Delta's approach [31], an effect size greater than 0.33 indicates a "large" difference between the two datasets. Therefore, we consider that AudioTest demonstrates a "large" improvement compared to existing approaches.

Answer to RQ1.1: *On natural datasets, AudioTest outperforms all the compared test prioritization methods across all subjects, with an average improvement of 12.63%∼54.58%.*

Table 6 presents the results for RQ1.2, which assess the efficiency of AudioTest. In our experiments, the feature generation process of AudioTest leverages multithreading to enhance efficiency. Specifically, the audio dataset is divided into equal-sized chunks, which are processed concurrently using multiple threads. The table exhibits the time required for feature generation with 8, 16, and 32 threads, respectively. Table 6 shows that the total runtime of AudioTest includes feature generation, model training, and prediction. When using 32 threads, the total runtime is approximately 1.5 minutes. With 16 threads, the runtime increases to 2.7 minutes. Finally, with 8 threads, the runtime reaches around 5.2 minutes. Although the total runtime of AudioTest is higher than that of the compared methods, AudioTest achieves an accuracy improvement of 12.63%∼54.58%. Considering the trade-off between effectiveness and efficiency, AudioTest remains a practical option.

Answer to RQ1.2: *The total runtime of AudioTest is around 1.5 minutes when using 32 threads, increasing to 2.7 minutes with 16 threads, and around 5.2 minutes with 8 threads. Considering the trade-off between effectiveness and efficiency, AudioTest remains a practical option.*

5.2 RQ2: Effectiveness of AudioTest on Noisy Test Inputs

Objectives: We evaluate AudioTest and the compared methods on noisy datasets.

Results: The experimental results of RQ2 are shown in Tables 7, 8, 9, 10, and 11. Table 7 compares the effectiveness of AudioTest and existing test prioritization methods on noisy datasets, showing that AudioTest consistently outperforms all other methods across different noise generation techniques. Table 8 details AudioTest's average effectiveness on noisy data and its improvement over other methods. The average APFD of AudioTest is 0.701, with an average improvement of 12.71% to 40.48% compared to other approaches. Table 9 presents the results of the statistical analysis on noisy datasets. We see that all the p-values between AudioTest and the compared approaches are less than 10^{-5}, which indicates that the improvement of AudioTest over existing approaches is statistically significant. Table 10 presents a comparison of AudioTest and existing methods using the PFD metric. The columns labeled "10%, 20%, 30%, 40%" in the table indicate the proportion of tests executed. The results demonstrate that AudioTest consistently outperforms the

Table 7. Effectiveness comparison on noisy data (APFD) with best results highlighted in gray

Noise Technique	Model	Approach					
		Random	DeepGini	Entropy	PCS	VanillaSM	AudioTest
Brown	CAMPPlus	0.512	0.622	0.621	0.621	0.622	0.732
	EcapaTdnn	0.498	0.666	0.664	0.664	0.667	0.743
	ERes2Net	0.495	0.623	0.622	0.621	0.623	0.703
	PANNS	0.507	0.633	0.630	0.631	0.633	0.739
	ResNetSE	0.499	0.635	0.632	0.633	0.635	0.733
	TDNN	0.496	0.641	0.638	0.642	0.642	0.729
Gaussian	CAMPPlus	0.501	0.597	0.597	0.596	0.598	0.701
	EcapaTdnn	0.506	0.582	0.581	0.579	0.582	0.703
	ERes2Net	0.515	0.585	0.586	0.579	0.584	0.705
	PANNS	0.502	0.607	0.606	0.605	0.608	0.712
	ResNetSE	0.506	0.629	0.626	0.628	0.629	0.715
	TDNN	0.506	0.621	0.618	0.619	0.620	0.698
Impulse	CAMPPlus	0.499	0.592	0.592	0.584	0.591	0.635
	EcapaTdnn	0.498	0.587	0.583	0.586	0.588	0.641
	ERes2Net	0.498	0.565	0.566	0.557	0.563	0.612
	PANNS	0.502	0.597	0.596	0.593	0.598	0.637
	ResNetSE	0.499	0.592	0.589	0.586	0.592	0.646
	TDNN	0.504	0.567	0.566	0.562	0.567	0.621
Sawtooth	CAMPPlus	0.496	0.571	0.565	0.568	0.571	0.647
	EcapaTdnn	0.495	0.619	0.609	0.623	0.623	0.703
	ERes2Net	0.497	0.574	0.570	0.572	0.575	0.627
	PANNS	0.497	0.591	0.583	0.597	0.596	0.649
	ResNetSE	0.498	0.588	0.584	0.585	0.589	0.665
	TDNN	0.498	0.609	0.605	0.609	0.611	0.679
Sine	CAMPPlus	0.499	0.712	0.701	0.714	0.715	0.781
	EcapaTdnn	0.488	0.727	0.720	0.726	0.729	0.778
	ERes2Net	0.502	0.649	0.641	0.653	0.653	0.722
	PANNS	0.491	0.718	0.711	0.717	0.721	0.751
	ResNetSE	0.504	0.672	0.664	0.673	0.675	0.751
	TDNN	0.492	0.678	0.668	0.681	0.682	0.726
Triangle	CAMPPlus	0.501	0.668	0.659	0.667	0.671	0.741
	EcapaTdnn	0.507	0.701	0.691	0.701	0.703	0.765
	ERes2Net	0.498	0.609	0.601	0.612	0.612	0.692
	PANNS	0.496	0.647	0.643	0.647	0.651	0.710
	ResNetSE	0.494	0.643	0.636	0.642	0.645	0.711
	TDNN	0.509	0.675	0.666	0.678	0.679	0.729
White	CAMPPlus	0.492	0.567	0.568	0.564	0.567	0.687
	EcapaTdnn	0.496	0.589	0.585	0.588	0.590	0.694
	ERes2Net	0.501	0.542	0.543	0.537	0.541	0.695
	PANNS	0.498	0.588	0.587	0.586	0.588	0.691
	ResNetSE	0.501	0.586	0.585	0.585	0.587	0.696
	TDNN	0.491	0.618	0.615	0.619	0.619	0.693

Table 8. Overall comparison results across all noisy subjects (APFD)

Approach	# Best cases	Average APFD	Improvement (%)
Random	0	0.499	40.48
DeepGini	0	0.621	12.88
Entropy	0	0.617	13.61
PCS	0	0.619	13.24
VanillaSM	0	0.622	12.71
AudioTest	78	0.701	-

Table 9. Statistical analysis on noisy datasets

	Random	DeepGini	Entropy	PCS	VanillaSM
p-value	2.14×10^{-8}	4.69×10^{-6}	9.32×10^{-7}	1.99×10^{-6}	1.54×10^{-6}
effect size	1.0	0.828	0.887	0.859	0.868

Table 10. Effectiveness comparison on noisy data (PFD)

Approach	# Best cases in PFD				Average PFD			
	10%	20%	30%	40%	10%	20%	30%	40%
Random	0	0	0	0	0.099	0.199	0.300	0.399
DeepGini	0	0	0	0	0.156	0.299	0.431	0.552
Entropy	0	0	0	0	0.154	0.294	0.426	0.547
PCS	0	0	0	0	0.151	0.296	0.429	0.551
VanillaSM	0	0	0	0	0.157	0.301	0.433	0.554
AudioTest	78	78	78	78	0.205	0.393	0.563	0.706

existing methods across different proportions of tests executed. Table 11 presents the experimental results on mixed noisy datasets. In Table 11, we see that, on mixed noisy datasets, AudioTest achieves the highest effectiveness compared to other approaches.

Table 11. Effectiveness comparison on mixed noisy data

Mixed Noisy Dataset	Approach					
	Random	DeepGini	Entropy	PCS	VanillaSM	AudioTest
Brown-Gaussian	0.503	0.619	0.618	0.618	0.621	0.717
Gaussian-Impulse	0.503	0.593	0.592	0.589	0.593	0.668
Impulse-Sawtooth	0.498	0.587	0.584	0.585	0.588	0.646
Sawtooth-Sine	0.501	0.642	0.635	0.643	0.645	0.706
Sine-Triangle	0.498	0.674	0.665	0.675	0.678	0.738
Triangle-White	0.499	0.619	0.614	0.618	0.621	0.708
Brown-Gaussian-Impulse	0.502	0.607	0.606	0.604	0.607	0.689
Gaussian-Impulse-Sawtooth	0.501	0.592	0.590	0.590	0.593	0.667
Impulse-Sawtooth-Sine	0.497	0.622	0.617	0.621	0.624	0.681
Sawtooth-Sine-Triangle	0.502	0.647	0.639	0.648	0.651	0.712
Sine-Triangle-White	0.499	0.643	0.638	0.643	0.646	0.722
Brown-Gaussian-Impulse-Sawtooth	0.501	0.603	0.603	0.601	0.604	0.682
Gaussian-Impulse-Sawtooth-Sine	0.499	0.617	0.613	0.616	0.619	0.687
Impulse-Sawtooth-Sine-Triangle	0.498	0.631	0.625	0.631	0.633	0.692
Sawtooth-Sine-Triangle-White	0.497	0.630	0.624	0.631	0.635	0.707

Table 12. Ablation study for feature contribution analysis

Approach	Dataset			Average
	Actor	FSD	UrbanSound	
AudioTest w/o TD	0.702	0.748	0.797	0.749
AudioTest w/o FD	0.697	0.745	0.795	0.746
AudioTest w/o PF	0.691	0.742	0.785	0.739
AudioTest w/o OF	0.683	0.665	0.764	0.704
AudioTest	0.729	0.766	0.836	0.776

Answer to RQ2: *On noisy datasets, AudioTest outperforms the compared test prioritization approaches, with an average improvement of 12.71%∼40.48%.*

5.3 RQ3: Feature Contribution Analysis

Objectives: We investigate whether each type of feature contributes to AudioTest's effectiveness.

Table 13. Statistical Analysis of Feature Contribution for AudioTest

	AudioTest w/o TD	AudioTest w/o FD	AudioTest w/o PF	AudioTest w/o OF
p-value	6.71×10^{-6}	7.54×10^{-6}	9.53×10^{-6}	8.31×10^{-7}
effect size	0.814	0.809	0.801	0.892

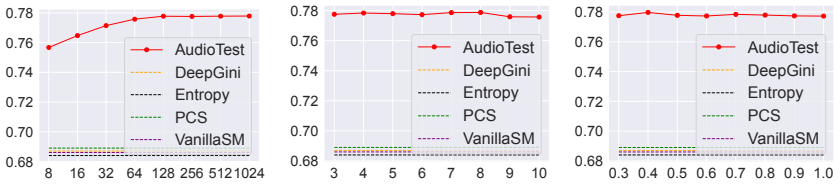
Results: Table 12 presents the ablation study results. 'W/o' means 'without,' so 'AudioTest w/o TD' refers to AudioTest without Time-Domain Features. We see that the original AudioTest is the most effective for each dataset. Removing any feature type reduces effectiveness. AudioTest's average effectiveness is 0.776 across all subjects. Removing OF decreases APFD by 0.072, PF by 0.037, FD by 0.03, and TD by 0.027. These results indicate that each feature type contributes to AudioTest. Table 13 presents the results of the statistical analysis of feature contribution for AudioTest. We see that all the p-values between AudioTest and AudioTest removing one type of feature are less than 10^{-5}. This indicates that removing any type of feature leads to a statistically significant decrease in AudioTest's effectiveness, demonstrating the contribution of each feature.

Answer to RQ3: *Each type of features generated by AudioTest contributes to its effectiveness.*

5.4 RQ4: Impact of Main Parameters on AudioTest

Objectives: We investigate to what extent the effectiveness of AudioTest is affected when the main parameters fluctuate. We focused on three parameters in AudioTest: max_depth (the maximum depth of each tree), colsample_bytree (the feature column sampling ratio when constructing each tree), and dimension (the size of features after transformation). Specifically, the parameter max_depth determines the complexity of the decision trees built by the model. The

colsample_bytree parameter specifies the proportion of features to be randomly sampled for training each tree. The dimension controls the size of each test's feature vector after feature transformation.



(a) dimension (b) max_depth (c) colsample_bynode

Fig. 5. Impact of main parameters in AudioTest

Results: Figure 5 illustrates the variation in the effectiveness of AudioTest when the values of the main parameters change. We see that as the parameters vary, the effectiveness of AudioTest remains stable, with the effectiveness variation range being less than 0.02. Specifically, the parameters evaluated include: **Dimension**: ranging from 8 to 1024, **Max Depth**: ranging from 3 to 10, and **Colsample Bynode**: ranging from 0.3 to 1.0. These results demonstrate that 1) AudioTest exhibits stable performance across a wide range of parameter settings, and 2) AudioTest consistently outperforms other test prioritization methods across different parameter settings.

Answer to RQ4: *AudioTest performs stably under different parameter settings. AudioTest consistently outperforms other test prioritization methods across various parameter settings.*

6 Threats to Validity

Internal Threats to Validity. Internal threats to validity mainly come from implementing the test prioritization approaches used for comparison. To mitigate it, we used the original implementations of the test prioritization approaches from their authors to minimize biases. Another threat arises from potential randomness in model training. To address this, we repeated all experiments 30 times [2], reported the average results, and conducted a statistical analysis. Another threat arises from the features generated within AudioTest. Specifically, whether the generated features can effectively capture the constructs they are intended to represent. To mitigate it, the selected features are drawn from established studies [1, 16, 18, 36, 41] in audio classification. In particular, time-domain and frequency-domain features were chosen to represent the physical properties of audio signals, while perceptual features were included to capture subjective human auditory perception. Output features were incorporated to provide insights into the model's prediction behavior.
External Threats to Validity. External threats primarily arise from the utilized audio datasets and models for evaluation. To mitigate it, we adopted different types of subjects, including both natural and noisy data, and applied seven different noise generation techniques for generating noisy tests. In total, we constructed 96 subjects for evaluation.

7 Related Work

7.1 Test Prioritization Techniques

To solve the labeling-cost problem, several works on DNN test prioritization have been proposed [10, 19, 44, 68, 75, 77]. Feng *et al.* [19] proposed DeepGini, which leverages model prediction confidence for prioritization. Weiss *et al.* [77] empirically investigated several existing DNN test prioritization methods, finding that some simple uncertainty-based approaches such as Vanilla Softmax perform equally well as DeepGini. Wang *et al.* [75] proposed PRIMA, which performed prioritization based

on intelligent mutation analysis. In traditional software testing [5, 13, 29, 55], Henard *et al.* [29] examined white-box and black-box prioritization techniques, finding that the differences between these approaches are small. Chen *et al.* [5] introduced LET, which utilizes a learning process to identify relevant program features and assess the bug-revealing potential of new tests. Shin *et al.* [66] proposed a diversity-aware mutation adequacy criterion for prioritization.

7.2 Deep Neural Network Testing

Besides test prioritization, DNN testing [3, 33, 34, 61–63, 69, 70, 84] also contains several other crucial aspects, such as test selection [6, 24, 43, 46, 79, 82] and adequacy measurement [15, 40, 49, 50, 60]. For test selection, Li *et al.* [46] proposed CES, which minimizes the cross-entropy between the selected set and the original set. Chen *et al.* [6] proposed PACE, which clusters all the tests and uses the MMD-critic algorithm [39] to perform prototype selection within each cluster. For adequacy measurement [15, 40, 49, 50, 60], Pei *et al.* [60] introduced neuron coverage to determine how well a test set covers the logic of a DNN model. Ma *et al.* [49] proposed DeepGauge, a set of coverage criteria to measure DNN test adequacy. Kim *et al.* [40] introduced surprise (i.e., the difference in the DL system's behavior between test and training data) to measure test input effectiveness.

8 Conclusion

To address the labeling cost issue for audio-type test inputs, we proposed AudioTest, a novel test prioritization approach specifically designed for audio test cases. The premise of AudioTest is that tests closer to misclassified samples are more likely to be misclassified. Following this premise, AudioTest generated four types of features based on the unique characteristics of audio data: time-domain, frequency-domain, perceptual, and output features. Then, utilizing a carefully designed feature transformation approach, AudioTest transforms the extracted features of all tests to bring misclassified tests closer in space, thereby ensuring the classification model utilized by AudioTest can better distinguish misclassified tests and correctly classified tests. Finally, AudioTest utilized a trained model to predict the misclassification probability of each test based on its transformed feature vector. AudioTest ranked all the tests based on their misclassification probability. We evaluated AudioTest based on 96 subjects, encompassing both natural and noisy datasets. The experimental results demonstrated that AudioTest outperforms all the compared methods, with an improvement of 12.63%~54.58% on natural datasets and 12.71%~40.48% on noisy datasets. Future work could focus on utilizing the tests selected by AudioTest to further fine-tune audio classification models and enhance their performance. Specifically, the community could explore the following research directions: 1) Investigating the effectiveness of AudioTest to enhance the performance of audio classification models through retraining. 2) Evaluating how to utilize potentially misclassified inputs to fine-tune the audio classification model, with the aim of enhancing its performance.

9 Data Availability

Our code and experiment data are made publicly available on Zenodo:

https://zenodo.org/records/13961855

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] Zrar Kh Abdul and Abdulbasit K Al-Talabani. 2022. Mel frequency cepstral coefficient and its applications: A review. *IEEE Access* 10 (2022), 122136–122158. https://doi.org/10.1109/ACCESS.2022.3223444

[2] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd international conference on software engineering*. 1–10. https://doi.org/10.1145/1985793.1985795

[3] Matteo Biagiola and Paolo Tonella. 2024. Boundary state generation for testing and improvement of autonomous driving systems. *IEEE Transactions on Software Engineering* (2024). https://doi.org/10.1109/TSE.2024.3420816

[4] Junjie Chen, Yiling Lou, Lingming Zhang, Jianyi Zhou, Xiaoleng Wang, Dan Hao, and Lu Zhang. 2018. Optimizing test prioritization via test distribution analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 656–667. https://doi.org/10.1145/3236024.3236053

[5] 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. https://doi.org/10.1109/TSE.2018.2889771

[6] 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. https://doi.org/10.1145/3394112

[7] 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. https://doi.org/10.1145/2939672.2939785

[8] Yafeng Chen, Siqi Zheng, Hui Wang, Luyao Cheng, Qian Chen, and Jiajun Qi. 2023. An enhanced res2net with local and global feature fusion for speaker verification. *arXiv preprint arXiv:2305.12838* (2023).

[9] Xueqi Dang, Yinghua Li, Wei Ma, Yuejun Guo, Qiang Hu, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 2024. Towards exploring the limitations of test selection techniques on graph neural networks: An empirical study. *Empirical Software Engineering* 29, 5 (2024), 112. https://doi.org/10.1007/s10664-024-10515-y

[10] 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* 33, 1 (2023), 1–40. https://doi.org/10.1145/3607191

[11] Nauman Dawalatabad, Mirco Ravanelli, François Grondin, Jenthe Thienpondt, Brecht Desplanques, and Hwidong Na. 2021. ECAPA-TDNN embeddings for speaker diarization. *arXiv preprint arXiv:2104.01466* (2021).

[12] Brecht Desplanques, Jenthe Thienpondt, and Kris Demuynck. 2020. Ecapa-tdnn: Emphasized channel attention, propagation and aggregation in tdnn based speaker verification. *arXiv preprint arXiv:2005.07143* (2020).

[13] 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. https://doi.org/10.1109/ICST.2013.27

[14] Daniel Di Nardo, Nadia Alshahwan, Lionel Briand, and Yvan Labiche. 2015. Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system. *Software Testing, Verification and Reliability* 25, 4 (2015), 371–396. https://doi.org/10.1002/stvr.1572

[15] 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. https://doi.org/10.1145/3576040

[16] Shlomo Dubnov. 2004. Generalization of spectral flatness measure for non-gaussian linear processes. *IEEE Signal Processing Letters* 11, 8 (2004), 698–701. https://doi.org/10.1109/LSP.2004.831663

[17] 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. https://doi.org/10.1109/32.988497

[18] Dan Ellis. 2007. Chroma feature analysis and synthesis. *Resources of laboratory for the recognition and organization of speech and Audio-LabROSA* 5 (2007).

[19] 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. https://doi.org/10.1145/3395363.3397357

[20] Eduardo Fonseca, Manoj Plakal, Daniel PW Ellis, Frederic Font, Xavier Favory, and Xavier Serra. 2019. Learning sound event classifiers from web audio with noisy labels. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 21–25. https://doi.org/10.1109/ICASSP.2019.8683158

[21] Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel PW Ellis, Xavier Favory, Jordi Pons, and Xavier Serra. 2018. General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline. *arXiv preprint arXiv:1807.09902* (2018).

[22] Liang Gao, Kele Xu, Huaimin Wang, and Yuxing Peng. 2022. Multi-representation knowledge distillation for audio classification. *Multimedia Tools and Applications* 81, 4 (2022), 5089–5112. https://doi.org/10.1007/s11042-021-11610-8

[23] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. 2017. Audio set: An ontology and human-labeled dataset for audio events. In *2017 IEEE international*

conference on acoustics, speech and signal processing (ICASSP). IEEE, 776–780. https://doi.org/10.1109/ICASSP.2017.7952261

[24] Antonio Guerriero, Roberto Pietrantuono, and Stefano Russo. 2024. DeepSample: DNN sampling-based testing for operational accuracy assessment. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12. https://doi.org/10.1145/3597503.3639584

[25] Hary Gunarto. 2024. Applications of AI-empowered electric vehicles for voice recognition in Asian and Austronesian languages. In *Artificial Intelligence-Empowered Modern Electric Vehicles in Smart Grid Systems*. Elsevier, 81–112. https://doi.org/10.1016/B978-0-443-23814-7.00004-3

[26] Shilpa Gupta, Varun Srivastava, and Deepika Kumar. 2024. Environment Sound Classification using stacked features and convolutional neural network. In *Proceedings of the 2024 Sixteenth International Conference on Contemporary Computing*. 42–50. https://doi.org/10.1145/3675888.3676028

[27] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*. 1–9. https://doi.org/10.1145/2648584.2648589

[28] Calum Heggan, Sam Budgett, Timothy Hospedales, and Mehrdad Yaghoobi. 2022. Metaaudio: A few-shot audio classification benchmark. In *International Conference on Artificial Neural Networks*. Springer, 219–230. https://doi.org/10.1007/978-3-031-15919-0_19

[29] 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. https://doi.org/10.1145/2884781.2884791

[30] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. 2017. CNN architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 131–135. https://doi.org/10.1109/ICASSP.2017.7952132

[31] Melinda R Hess and Jeffrey D Kromrey. 2004. Robust confidence intervals for effect sizes: A comparative study of Cohen'sd and Cliff's delta under non-normality and heterogeneous variances. In *annual meeting of the American Educational Research Association*, Vol. 1. Citeseer.

[32] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.

[33] 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. https://doi.org/10.1145/3460319.3464825

[34] 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. https://doi.org/10.1109/ICST46399.2020.00018

[35] jiaaro. 2024. *pydub*. https://github.com/jiaaro/pydub/tree/master

[36] Dan-Ning Jiang, Lie Lu, Hong-Jiang Zhang, Jian-Hua Tao, and Lian-Hong Cai. 2002. Music type classification by spectral contrast feature. In *Proceedings. IEEE international conference on multimedia and expo*, Vol. 1. IEEE, 113–116. https://doi.org/10.1109/ICME.2002.1035731

[37] R Juzenaite. 2019. Security vulnerabilities of voice recognition technologies.

[38] Kaggle. 2024. Kaggle - The Machine Learning and Data Science Community. https://www.kaggle.com.

[39] 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).

[40] 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. https://doi.org/10.1109/ICSE.2019.00108

[41] Anssi Klapuri and Manuel Davy. 2007. Signal processing methods for music transcription. (2007).

[42] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D Plumbley. 2020. Panns: Large-scale pretrained audio neural networks for audio pattern recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 28 (2020), 2880–2894. https://doi.org/10.1109/TASLP.2020.3030497

[43] Taeckyung Lee, Sorn Chottananurak, Taesik Gong, and Sung-Ju Lee. 2024. AETTA: Label-Free Accuracy Estimation for Test-Time Adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 28643–28652.

[44] Yinghua Li, Xueqi Dang, Lei Ma, Jacques Klein, Yves Le Traon, and Tegawendé F Bissyandé. 2024. Test input prioritization for 3d point clouds. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 1–44. https://doi.org/10.1145/3643676

[45] Zheng Li, Mark Harman, and Robert M Hierons. 2007. Search algorithms for regression test case prioritization. *IEEE Transactions on software engineering* 33, 4 (2007), 225–237. https://doi.org/10.1109/TSE.2007.38

[46] 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. https://doi.org/10.1145/3338906.3338930

[47] Steven R Livingstone and Frank A Russo. 2018. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. *PloS one* 13, 5 (2018), e0196391. https://doi.org/10.1371/journal.pone.0196391

[48] Li Lu, Jiadi Yu, Yingying Chen, and Yan Wang. 2020. Vocallock: Sensing vocal tract for passphrase-independent user authentication leveraging acoustic signals on smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 2 (2020), 1–24.

[49] 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. https://doi.org/10.1145/3238147.3238202

[50] 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. https://doi.org/10.1109/ISSRE.2018.00021

[51] 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. https://doi.org/10.1145/3417330

[52] Angel Mario Castro Martinez, Constantin Spille, Jana Roßbach, Birger Kollmeier, and Bernd T Meyer. 2022. Prediction of speech intelligibility with DNN-based performance measures. *Computer Speech & Language* 74 (2022), 101329. https://doi.org/10.1016/j.csl.2021.101329

[53] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. 2015. librosa: Audio and music signal analysis in python. In *SciPy*. 18–24.

[54] Patrick E McKnight and Julius Najab. 2010. Mann-Whitney U Test. *The Corsini encyclopedia of psychology* (2010), 1–1.

[55] Cu Nguyen, Paolo Tonella, Tanja Vos, Nelly Condori, Bilha Mendelson, Daniel Citron, and Onn Shehory. 2014. Test prioritization based on change sensitivity: an industrial case study. *Technical Report Series/Department of Information and Computing Sciences Utrecht University* UU-CS-2014-012 (2014).

[56] 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. https://doi.org/10.1155/2021/4832864

[57] Taikang Ning, James Ning, Nikolay Atanasov, and Kai-Sheng Hsieh. 2012. A fast heart sounds detection and heart murmur classification algorithm. In *2012 IEEE 11th International Conference on Signal Processing*, Vol. 3. IEEE, 1629–1632.

[58] Tanzeem Bin Noor and Hadi Hemmati. 2015. A similarity-based approach for test case prioritization using historical failure data. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 58–68.

[59] 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).

[60] 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. https://doi.org/10.1145/3361566

[61] 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. https://doi.org/10.1109/ASE51524.2021.9678764

[62] 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. https://doi.org/10.1145/3368089.3409730

[63] Vincenzo Riccio and Paolo Tonella. 2023. When and why test generators for deep learning produce invalid inputs: an empirical study. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1161–1173. https://doi.org/10.1109/ICSE48619.2023.00104

[64] Lior Rokach and Oded Maimon. 2005. Decision trees. *Data mining and knowledge discovery handbook* (2005), 165–192. https://doi.org/10.1007/0-387-25465-X_9

[65] J. Salamon, C. Jacoby, and J. P. Bello. 2014. A Dataset and Taxonomy for Urban Sound Research. In *22nd ACM International Conference on Multimedia (ACM-MM'14)*. Orlando, FL, USA, 1041–1044. https://doi.org/10.1145/2647868.2655045

[66] 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. https://doi.org/10.

1002/stvr.1695

[67] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. 2018. X-vectors: Robust dnn embeddings for speaker recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 5329–5333. https://doi.org/10.1109/ICASSP.2018.8461375

[68] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 359–371. https://doi.org/10.1145/3377811.3380353

[69] Zohdinasab Tahereh, Vincenzo Riccio, Tonella Paolo, et al. 2023. DeepAtash: Focused Test Generation for Deep Learning Systems. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*. https://doi.org/10.1145/3597926.3598109

[70] Deepak-George Thomas, Matteo Biagiola, Nargiz Humbatova, Mohammad Wardat, Gunel Jahangirova, Hridesh Rajan, and Paolo Tonella. 2024. muPRL: A Mutation Testing Pipeline for Deep Reinforcement Learning based on Real Faults. *arXiv preprint arXiv:2408.15150* (2024).

[71] Stephen W Thomas, Hadi Hemmati, Ahmed E Hassan, and Dorothea Blostein. 2014. Static test case prioritization using topic models. *Empirical Software Engineering* 19 (2014), 182–212. https://doi.org/10.1007/s10664-012-9219-7

[72] 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. https://doi.org/10.1109/ICSM.2006.74

[73] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008). http://jmlr.org/papers/v9/vandermaaten08a.html

[74] Hui Wang, Siqi Zheng, Yafeng Chen, Luyao Cheng, and Qian Chen. 2023. Cam++: A fast and efficient network for speaker verification using context-aware masking. *arXiv preprint arXiv:2303.00332* (2023).

[75] 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. https://doi.org/10.1109/ICSE43902.2021.00046

[76] Zhengyuan Wei, Haipeng Wang, Imran Ashraf, and WK Chan. 2022. Predictive mutation analysis of test case prioritization for deep neural networks. In *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 682–693. https://doi.org/10.1109/QRS57517.2022.00074

[77] 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. https://doi.org/10.1145/3533767.3534375

[78] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*. Springer, 408–426. https://doi.org/10.1007/978-3-319-89960-2_22

[79] Zhuo Wu, Zan Wang, Junjie Chen, Hanmo You, Ming Yan, and Lanjun Wang. 2024. Stratified random sampling for neural network test input selection. *Information and Software Technology* 165 (2024), 107331. https://doi.org/10.1016/j.infsof.2023.107331

[80] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120. https://doi.org/10.1002/stv.430

[81] 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. https://doi.org/10.1145/1572272.1572296

[82] Lin Zhao, Tianchen Zhao, Zinan Lin, Xuefei Ning, Guohao Dai, Huazhong Yang, and Yu Wang. 2024. FlashEval: Towards Fast and Accurate Evaluation of Text-to-image Diffusion Generative Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16122–16131. https://doi.org/10.1109/CVPR52733.2024.01526

[83] Zifeng Zhao, Ding Pan, Junyi Peng, and Rongzhi Gu. 2022. Probing Deep Speaker Embeddings for Speaker-related Tasks. *arXiv preprint arXiv:2212.07068* (2022). https://arxiv.org/abs/2212.07068

[84] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2023. Efficient and effective feature space exploration for testing deep learning systems. *ACM Transactions on Software Engineering and Methodology* 32, 2 (2023), 1–38. https://doi.org/10.1145/3544792

Received 2024-10-23; accepted 2025-03-31