LongTest: Test Prioritization for Long Text Files

Xueqi Dang, Yinghua Li, Wendkûuni C. Ouédraogo, Maxime Cordy, Mike Papadakis,
Jacques Klein, Tegawendé F. Bissyandé and Yves Le Traon

Abstract—Text classification is a fundamental task in natural language processing (NLP) that involves assigning predefined labels to text files. While existing research has primarily focused on short texts (e.g., social media posts), the classification of long texts, such as legal documents and medical records, has gained increasing attention due to the rising demand for analyzing complex and information-rich documents. However, testing long text classification systems poses unique challenges due to the high manual labeling costs associated with annotating test inputs (long text files). Test input prioritization has emerged as a promising strategy to mitigate this challenge, which prioritizes potentially misclassified tests during the early stages of the testing process. By labeling these inputs earlier, it accelerates the debugging process and thus improves the efficiency of DNN testing. However, applying existing methods to prioritize long text files is constrained by several factors: 1) Confidence-based methods fail to leverage the rich semantic content of the test inputs (long text files); 2) mutation-based techniques are specifically designed for short texts, introducing small mutations that have negligible impacts on long text files; and 3) coverage-based methods are computationally expensive and less effective. To overcome these limitations, we propose LongTest, a novel test input prioritization approach specifically designed for long text files. LongTest introduces: **1) an embedding generation mechanism** specifically designed to enhance the capture of information from the entire long text file, and **2) contrastive learning** that enables more effective differentiation between misclassified and correctly classified samples, which optimizes the test prioritization process. The results demonstrate that LongTest outperforms all the compared test prioritization approaches. Specifically, compared with the existing test prioritization methods (i.e., DeepGini, VanillaSM, PCS, Entropy, and DATIS), LongTest achieves an average improvement of 9.61%∼16.31%.

Index Terms—Test Input Prioritization, Deep Neural Network, Labeling

1 Introduction

Text classification is a foundational problem in natural language processing (NLP) [1], which focuses on assigning pre-defined labels to text files. While traditional research [1], [2] has primarily targeted short texts (e.g., social media posts, emails, and product reviews), the increasing complexity of real-world applications demands more advanced methods capable of analyzing long text files, such as legal documents, scientific literature, and technical reports.

Compared to short texts, long texts are characterized by their extended length, complex hierarchical structure, and rich semantic diversity, significantly distinguishing them from short texts [3]. The classification of long texts holds critical importance across various domains, such as medical report classification [4], legal document categorization [5], and research topic classification [6], [7]. For example, in the medical domain [8], classification models can be utilized to categorize patients' medical history records (which are long text files) into different disease types, thereby facilitating accurate diagnoses and improving treatment planning.

However, in such critical scenarios, misclassifications can lead to severe consequences. For example, if a patient's record is misclassified, it could result in misdiagnosis or inappropriate treatment, delaying the correct medical process and potentially worsening the patient's condition. Therefore, ensuring the accuracy and reliability of such long-text classification systems is essential. Testing is a fundamental practice for ensuring the quality of DNN-based systems [9]. However, there is a core challenge in testing long text classification models, which is the labeling cost problem (i.e., labeling all test inputs to verify the correctness of these models can be expensive). This challenge primarily arises from three crucial aspects: 1) Manual annotation remains a mainstream method for labeling, making the process labor-intensive and time-consuming. 2) Test sets for long text classification can be large-scale, increasing the labeling workload. 3) Domain-specific knowledge is frequently required in certain fields to accurately label long text files. For example, annotating medical history records can require expertise from medical professionals, further increasing the labelling cost.

To alleviate the labelling cost problem, one intuitive and effective approach is test input prioritization [9], [10], [11], which focuses on identifying and prioritizing test inputs that are more likely to be misclassified by the DNN model, as these inputs are more likely to expose faults in the model. This approach enables the detection of more fault-revealing test inputs within a limited timeframe. Labeling these inputs earlier accelerates the debugging process and thus improves the efficiency of DNN testing.

In the literature, several test prioritization methods have been proposed to enhance the efficiency of DNN testing. These approaches can be broadly categorized into three classes: coverage-based [12], [13], confidence-based [9], [10], and mutation-based approaches [11]. Coverage-based methods adapt traditional software testing techniques to DNN testing by leveraging neuron coverage metrics for test pri-

- *X. Dang, W. Ouédraogo, M. Cordy, M. Papadakis, J. Klein and T. Bissyandé and Y. Traon are with University of Luxembourg.*
  Y. Li is with Nanjing University of Science and Technology.
  E-mail: xueqi.dang@uni.lu, yinghua.li@njust.edu.cn, wendkuuni.ouedraogo@uni.lu, maxime.cordy@uni.lu, michail.papadakis@uni.lu, jacques.klein@uni.lu, tegawende.bissyande@uni.lu, yves.letraon@uni.lu
- *Yinghua Li is the corresponding author.*

oritization. Confidence-based methods focus on prioritizing test inputs where the model exhibits lower confidence in its predictions. For example, DeepGini [9] uses the Gini score to quantify the model's prediction confidence and prioritizes inputs with higher uncertainty. Mutation-based methods [11] introduce novel mutation operators and utilize mutation results to guide test prioritization.

However, although these approaches have demonstrated effectiveness in certain cases, they suffer from the following limitations when applied to the context of long text files:

- Confidence-based approaches rely solely on the output probabilities generated by the final layer of the model, which represent a compressed summary of the model's confidence in its predictions. They do not utilize the information contained within the original input data, especially the rich semantic and hierarchical structure of long text files, which, as a result, limits their effectiveness in prioritizing long text files.
- Mutation-based approaches are typically designed for short text. Their proposed mutation operators usually introduce small-scale mutations, such as altering a few characters within a word. However, long text typically consists of extended content and numerous words, making the impact of such slight mutations negligible, which makes mutation-based methods unsuitable for test prioritization in this context. Moreover, the newly proposed mutation operators of mutation-based approaches are specifically designed for DNNs and are not applicable to the classical machine learning component of the composite model used in this study. As a result, mutation-based methods are not suitable in this context.
- Coverage-based test prioritization methods have been demonstrated to be less effective and more computationally expensive compared to confidence-based methods [9]. Moreover, coverage-based methods prioritize tests based on the activation coverage of neurons in DNN models. However, the long text classification models in our paper are combo models (i.e., composite models consisting of an embedding model and a classical machine learning model). The classical machine learning components (e.g., Random Forest [14], Logistic Regression [15], and XGBoost [16]) do not contain neural network structures. As a result, coverage-based test prioritization methods are not applicable to the composite models used in this work and are therefore not suitable in this context.

In this paper, we propose LongTest (**Long** text file-oriented **Test** Input Prioritization), a learning-based test input prioritization approach specifically designed for prioritizing long text files. The core idea is that we utilize *contrastive learning* to bring misclassified tests closer together in the space while pushing misclassified and correctly classified tests farther apart. This enables better differentiation between misclassified and correctly classified tests, thereby achieving more effective test prioritization.

Specifically, LongTest performs test prioritization based on four steps: **1) Text Splitting and Transforming.** For each test input (a long text file) in the test set, LongTest divides it into smaller chunks, converts each chunk into an embedding, and concatenates all chunk embeddings to generate a final comprehensive embedding vector. This process aims to enhance the capture of information from the entire long text, particularly because embedding algorithms can have input token limitations (i.e., The embedding algorithms are constrained by the input text length and cannot capture information exceeding this limit). **2) Dimensionality Reduction.** LongTest applies Principal Component Analysis (PCA) [17] to reduce the dimensionality of the embedding vector of each test, aiming to decrease the overall runtime and improve efficiency while retaining the essential characteristics of the original data.

3) Contrastive Learning. We construct a new training dataset from the original training set to train the contrastive learning model. The embedding vector of each test is then fed into the trained contrastive learning model. As a result, the original embedding vector of each test is transformed into a contrastive vector. **4) Ranking.** LongTest employs a trained ranking model to estimate the misclassification probability for each test based on its contrastive vector and ranks all tests according to these probability values.

In the above process, labeled data are only required for the training set to construct the contrastive learning model and the ranking model. All test inputs in the test set are unlabeled, and LongTest ranks them according to the predicted misclassification scores.

LongTest has the following notable advantages when applied to prioritize long text files:

- **Leveraging Long Text Characteristics** Unlike confidence-based methods that solely rely on output probabilities from the model's final layer, LongTest incorporates rich semantic information from the entire long text file. Specifically, LongTest processes long text files by splitting them into chunks, generating embedding vectors, and applying Principal Component Analysis (PCA) for dimensionality reduction. This process preserves essential information of long text files while reducing the total running time and improving efficiency.
- **Enhancing Differentiation via Contrastive Learning** LongTest leverages contrastive learning to bring misclassified test samples closer together in the feature space while pushing misclassified and correctly classified samples further apart. This approach enables LongTest to better distinguish between misclassified and correctly classified samples, thereby improving the prioritization effectiveness.
- **Specifically Designed for Long Text** LongTest is specifically designed for prioritizing long text files, fully accounting for the characteristics of their extended length. Traditional mutation-based prioritization methods are designed for short texts, with mutation operators proposed specifically for them, making such methods unsuitable for long text files.

The core novelty of LongTest is summarized as follows:

- **First use of contrastive learning for test prioritization.** LongTest is the first approach to introduce contrastive learning into the field of test input prioritization for text classification. By leveraging contrastive learning to better separate misclassified from correctly classified samples in the embedding space, LongTest achieves significant performance improvement over existing test prioritization methods, including the state-of-the-art approach DATIS.

- **Specific investigation of long-text splitting strategies.** LongTest specifically explores splitting strategies focused on the characteristics of long texts, including fixed-size chunking, semantic-based splitting, paragraph-based splitting, and structure-aware splitting.
- **Comprehensive analysis of framework components.** We conduct in-depth studies on how key components and parameters influence the effectiveness of LongTest. This includes analyzing the impact of embedding model choice, embedding dimensionality, loss functions (e.g., Euclidean, cosine, Manhattan, SimCSE), and main parameters on LongTest's performance.

LongTest demonstrates broad applicability across various scenarios. By ranking inputs, LongTest ensures that limited resources are directed first toward the samples that are prioritized higher, which are most likely to be misclassified. In the medical domain, for example, models could be utilized to categorize patient history records into different disease types. Instead of treating all records equally, LongTest could identify and prioritize those records that the model is more likely to misclassify, thereby moving these potentially misclassified samples into the top portion of the ranked list for expert review. A similar situation occurs in legal document classification. If a compliance-sensitive file were incorrectly predicted as routine, the result could be costly delays or even legal consequences. By ranking documents by their likelihood of misclassification, LongTest could enable legal experts to review these high-risk cases promptly.

To assess the effectiveness of LongTest, we performed a comprehensive experimental evaluation using 45 subjects comprising three datasets and 15 combo models. Most of these combo models consist of an embedding model paired with a machine learning model. We compared LongTest against several existing test prioritization approaches that have demonstrated effectiveness in prior research [9], [10]. Our experimental results show that LongTest significantly outperforms the existing test prioritization methods. Compared with existing approaches, including DeepGini, VanillaSM, PCS, Entropy, and DATIS, LongTest achieves an average improvement ranging from 9.61% to 16.31%. In addition, when compared with the random baseline, LongTest achieves an improvement of 70.86%.

To facilitate further research, we have made our dataset, results, and tools publicly available on Zenodo[1].

Our work has the following major contributions:

- **Approach** We proposed LongTest, a test prioritization approach specifically designed for prioritizing long text files. To this end, we designed an embedding generation method tailored for long texts and applied contrastive learning to enhance the effectiveness of test prioritization.
- **Study** We evaluated LongTest using 45 subjects, including three datasets and 15 combo models. The experimental results demonstrate that LongTest outperforms all the compared test prioritization approaches, with an average improvement of 9.61%∼16.31%.
- **Performance Analysis** We investigate the impact of various parameters and components of LongTest on its performance, including the number of chunks (RQ2), embedding models (RQ3), dimensionality (RQ4), and main parameters (RQ5). Additionally, we conduct an ablation study (RQ6) to analyze the contribution of each individual component to the effectiveness of LongTest. We further extend our analysis to examine the impact of different splitting methods (RQ7), different loss functions (RQ8), and how LongTest can be leveraged to enhance downstream model performance (RQ9). Moreover, we investigate the impact of different ranking models on the effectiveness of LongTest (RQ10), evaluate the performance of LongTest on both long-text and short-text documents (RQ11), and analyze how different target model accuracy levels affect the effectiveness of LongTest (RQ12).

1. https://zenodo.org/records/14851892

2 Background

2.1 Deep Neural Networks

Deep Neural Networks (DNNs) consist of multiple layers, each containing numerous interconnected units known as neurons [18]. These neurons are connected by links, each link assigned a specific weight that is optimized during the model's training process. The training data guides the adjustment of these weights, enabling the network to learn and generalize patterns across various tasks. Text classification [19] is a common application of DNNs, focusing on categorizing textual data into predefined classes based on its content. It is widely used in fields like sentiment analysis [20], spam detection [21], [22], and medical record analysis [23]. Text classification tasks can vary depending on text length and complexity. Based on the existing study, a long document is defined as text containing more than 512 BERT tokens [24]. This is because the standard BERT input limit is 512 tokens, so any text exceeding this threshold is considered "long".

Short texts, such as social media posts or product reviews, are typically brief, with a small number of characters and straightforward content. Conversely, long texts, including news articles or legal documents, contain extensive information with complex structures, often spanning multiple topics and sentiments. The classification of long texts is critically important across various domains, such as medical report classification [4] and legal document categorization [5]. In such safety-critical scenarios, misclassifications can lead to severe consequences, making sufficient testing necessary and essential. A major challenge in testing long-text classification models lies in the high cost of labeling, as annotating all test inputs to verify model correctness can be expensive. In this paper, we deal with this challenge by proposing a novel test prioritization approach, called LongTest, which leverages the unique characteristics of long-text test inputs to perform test prioritization.

2.2 Contrastive Learning

Contrastive learning [25] learns effective data representations by pulling similar samples closer in feature space and pushing dissimilar samples apart. In recent years, contrastive learning has shown notable performance across various fields, especially with applications in image, text, and multi-modal data [26]. The primary objective of contrastive learning is to identify a parametric function $f_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^d$

that maps an input image $\mathbf{x} \in \mathbb{R}^D$ to a feature representation $\mathbf{z} = f_\theta(\mathbf{x}) \in \mathbb{R}^d$. This representation $\mathbf{z}$ in the feature space is designed to capture the semantic similarities between input samples. To accomplish this objective, contrastive learning leverages a **contrastive loss** function to optimize the network f_θ. The contrastive loss is designed to draw semantically similar samples closer together in the feature space while pushing apart dissimilar samples. This method enables the model to learn discriminative features that effectively distinguish between different categories (between positive samples and negative samples).

In our study, we propose the LongTest method, which leverages contrastive learning to enhance test prioritization effectiveness on long text files. Specifically, we define misclassified tests (those incorrectly predicted by the DNN model) as positive samples and correctly classified tests as negative samples. By training the contrastive learning model in this manner, LongTest aims to better distinguish between misclassified and correctly classified cases, thereby improving the model's ability to predict the probability of misclassification for a given test input. This can ultimately enhance the effectiveness of test prioritization.

2.3 Test Input Prioritization for DNNs

Test prioritization aims to identify and prioritize potentially misclassified test cases earlier in the testing process [9], [27], [28], [29], [30], [31], [32]. After prioritization, the test set is ranked such that tests more likely to be misclassified (i.e., fault-revealing tests) are prioritized higher. Based on this well-ranked test set, testers can select a subset of fault-revealing tests for early labeling, thereby reducing labeling costs [33]. Moreover, even if the labeling process is halted at an arbitrary point due to resource limitations, it can still achieve maximum benefit from human efforts [9].

Given a model M to test and a test suite T, the goal of test input prioritization is to systematically rank the test cases in T so that, when testing is halted at a certain point, the executed test cases from the prioritized set can reveal as many faults as possible. This approach enables the identification of bug-revealing test inputs within a limited timeframe, facilitating earlier debugging and improving the efficiency of DNN testing.

The existing DNN test prioritization approaches can be broadly classified into three main categories: coverage-based [12], [13], confidence-based [9], [10], [34], and mutation-based approaches [11], [35]. Coverage-based approaches adapt traditional software testing methods to DNN testing by leveraging neuron coverage metrics for prioritizing test inputs. Confidence-based approaches, on the other hand, prioritize test cases based on the model's prediction confidence, under the assumption that test cases with lower confidence are more likely to be misclassified. For example, DeepGini [9] calculates the Gini score to quantify the model's confidence for each test case and ranks the test cases within the test set based on these scores. Mutation-based approaches [11] introduce novel mutation operators, including model and input mutation operators, and utilize the mutation results for test prioritization.

However, although the above approaches have demonstrated effectiveness in certain cases, they face limitations when applied to long text files: 1) Coverage-based methods have been shown to be less effective and more computationally expensive compared to confidence-based approaches [9]. 2) Confidence-based approaches depend solely on output probabilities from the model's final layer. They do not use the rich semantic and hierarchical structure of test inputs (long text files) for test prioritization, limiting their effectiveness. 3) Mutation-based methods, typically designed for short text, focus on small-scale mutations, such as modifying a few characters of a word. These small mutations have negligible impact on long text files due to their extended length, making mutation-based approaches unsuitable for such cases.

To address the above limitations, in this paper, we propose LongTest, which leverages the unique characteristics of long text files for test prioritization. Section 5.1 demonstrates the effectiveness of LongTest.

LongTest has the following advantages in the context of test prioritization for long text files:

❶ **Leveraging the characteristics of long text files themselves.** LongTest leverages the rich semantic information from the entire long text file (test inputs). It splits long texts into chunks, generates embeddings, and applies PCA for dimensionality reduction, which preserves essential information while reducing the computation time of test prioritization. Existing confidence-based methods rely solely on the output probabilities from the model's final layer, without leveraging the rich information inherent in the long text itself.

❷ **Using contrastive learning to better distinguish between misclassified and correctly classified test inputs.** One of the core components of LongTest, contrastive learning, is designed to better distinguish between misclassified and correctly classified test inputs. After contrastive learning, misclassified test samples are brought closer together in the feature space, which improves the effectiveness of test prioritization.

❸ **Specifically designed for "long" texts.** LongTest is specifically designed for "long" text files, which are characterized by extended length and complex structure. Its core components, such as text preprocessing and dimensionality reduction, are designed specifically for long texts. In contrast, traditional mutation-based prioritization methods are primarily effective for short texts. These methods typically apply character-level mutations, such as shuffling letters (e.g., machine → macihne) or repeating characters (e.g., subfield → subfieeld) in a word. Such mutations tend to have a negligible impact on long text files.

Industrial Adoption of Test Prioritization Beyond academic studies, there is growing evidence that test input prioritization has practical value in industry. For example, PRIMA [11] was successfully applied to the testing of real-world autonomous-vehicle models in a large motor company. The results demonstrated that prioritization can effectively reduce labeling costs while identifying more bug-revealing test inputs within limited resources. Similarly, an industrial case study at Hyundai Motor Company showed that applying test input selection metrics for DNN-based semantic segmentation can reduce manual labeling costs by

30–50% with negligible accuracy loss [36]. Such evidence highlights the practical importance of advancing test prioritization methods, motivating our work on LongTest.

Importance of Test Prioritization in the Long-Text Domain. While prior work has primarily focused on test input prioritization in the context of traditional DNNs, test prioritization is also of high importance in the domain of long-text classification. Below, we elaborate on this motivation in detail. First, long texts are widely used in critical application domains such as healthcare and law [4], [5]. For example, in the medical field [8], classification models can be employed to categorize patients' medical history records (which are often lengthy and complex) into different disease categories, thereby supporting accurate diagnoses and effective treatment planning. In such scenarios, if misclassifications occur, they may lead to serious consequences, such as misdiagnosis or inappropriate treatment, delaying proper medical intervention and potentially worsening patient outcomes.

However, identifying misclassified inputs in these domains typically requires substantial labeling effort. This is because labeling long-text data is time-consuming due to its length and complexity, and it often requires domain-specific expertise. For instance, annotating patients' medical history records could require trained medical professionals, which further increases the labeling cost. Although no existing studies have specifically investigated test input prioritization for long texts, the need is evident given the growing application of long-text classification in high-stakes domains.

In such cases, test prioritization methods specifically for long text files can be used to effectively prioritize test inputs that are more likely to be misclassified. By ranking these high-risk samples earlier, it allows human annotators to focus their limited labeling resources on the most error-prone test cases first, thereby improving fault detection efficiency and reducing the overall labeling cost.

3 Approach

In this section, we introduce the specific methodology of LongTest. Section 3.1 presents an overview of the construction of LongTest (how LongTest is built). Sections 3.2 to 3.5 detail each step of the LongTest method. Section 3.6 demonstrates the usage of LongTest (how to use LongTest).

3.1 Overview

In this paper, we propose a novel test prioritization approach called LongTest. LongTest is specifically designed for prioritizing long text files. Specifically, the input to LongTest is an unlabelled test set T and a text classification model M to be evaluated. LongTest will output a prioritized test set T', where tests that are more likely to be misclassified are prioritized higher. In this section, we present an overview of how LongTest is constructed, as illustrated in Figure 1. As shown in the figure, the construction of LongTest mainly consists of four steps: 1) Text Preprocessing and Dimensionality Reduction, 2) Constructing Positive and Negative Pairs, 3) Training a Contrastive Learning Model, and 4) Training a Ranking Model for Prioritization. Detailed explanations of each step can be found in Sections 3.2 to 3.5.

3.1.1 Construction Pipeline

❶ **Step1: Text Preprocessing and Dimensionality Reduction** Given the training set R, for each long text-type sample $l \in R$, we first preprocess it. This preprocessing includes three key steps: 1) splitting the long text l into several chunks, 2) transforming each chunk into embeddings, and 3) concatenating the embeddings of all chunks to obtain a final embedding vector, denoted as V_l. Next, for each sample l, we perform Principal Component Analysis (PCA) [17] to reduce the dimensionality of its vector V_l and obtain a reduced-dimensional vector V_l^{PCA}.

❷ **Step2: Constructing Positive and Negative Pairs** After obtaining the final dimension-reduced embedding vector of each sample in R, we use these vectors to create pairs (e.g., positive & negative pairs). Here, "positive" represents samples that are misclassified by the model, while "negative" represents samples that are correctly classified by the model. Since we know the ground truth for each sample in R, we can simply input each sample in R into the text classification model M, compare its prediction with the ground truth, and annotate each sample as misclassified (positive) or correctly classified (negative). The generated pairs are then passed to the Base Network in Step 3 to train the contrastive learning model.

❸ **Step3: Training Contrastive Learning Model** In this step, we use the paired data from the previous step to train the contrastive learning model. After training, this contrastive learning model will bring the embedding vectors of misclassified tests closer to each other in space and make the embedding vectors of correctly classified tests closer to each other. Additionally, it will make the misclassified tests farther away from the correctly classified tests. This is intended to enable the subsequent ranking model to better predict whether a test will be misclassified or correctly classified based on its vector.

Specifically, after training, if we input the embedding vector V_t^{PCA} of a new test t into the contrastive learning model, it will output a transformed contrastive vector V_t^{cont}. The contrastive vector V_t^{cont} can better reflect how likely a test t will be misclassified.

❹ **Step4: Training Ranking Model for Prioritization** In the final step, we trained a ranking model for test prioritization. This model predicts whether a sample will be misclassified based on its contrastive vector, where a misclassified sample is labelled as "1" and a correctly classified sample as "0". Once the model was trained, we made some adjustments to it. Specifically, given an input (the contrastive vector of a test), we make the model output the intermediate value instead of the binary label (0 or 1). This value is a probability value indicating the likelihood that the test will be misclassified. We refer to it as the "misclassification score". The higher this score, the more likely the test will be misclassified.

3.1.2 Final tool for test prioritization

After the above process, the trained LongTest tool can be directly used for test prioritization (regarded as the final tool from the pipeline for the specific domain). Specifically, this trained LongTest model can be repeatedly used for test prioritization within the same domain without retraining.

The detailed procedure for using LongTest to rank tests can be found in Section 3.6.

3.1.3 Calibration for new domains

In practice, LongTest can be adapted to different application domains through a calibration process. The calibration process follows the same pipeline described in Section 3.1. First, the training samples from the new domain are preprocessed by splitting long texts into chunks, generating embedding vectors, and applying dimensionality reduction. Second, based on the ground-truth labels in the training set, LongTest constructs positive and negative sample pairs according to whether each sample is misclassified by the target classification model. These pairs are then used to train the contrastive learning model. After the contrastive learning model is trained, LongTest transforms the embedding vectors of the training samples into contrastive vectors. Using these vectors as features and the misclassification labels as targets, LongTest trains the ranking model to estimate the probability that a test input will be misclassified.

Once this calibration process is completed, the trained LongTest model can be repeatedly used for test prioritization within the same domain without requiring further retraining. When another new domain is encountered, the same calibration procedure can be applied again using the training dataset from that domain to adapt LongTest accordingly.

3.2 Step 1: Text Preprocessing and Dimensionality Reduction

Given the training set R and the text classification model M to be evaluated, for each long-text sample $l \in R$, we first performed text preprocessing and dimensionality reduction on it. In the following, we provide a detailed explanation of each specific procedure, along with the relevant formulas.

3.2.1 Chunk-based Text Splitting

Given the long text sample $l \in R$, we split l into several chunks, as described in Formula 1.

$$l = [c_1, c_2, \ldots, c_n] \tag{1}$$

where c_i represents the i_{th} chunk. n is the total number of chunks.

Specifically, we split the text sequentially, and each chunk contains approximately the same number of tokens. The number of tokens per chunk is calculated as follows.

$$t_c = \frac{T}{N} \tag{2}$$

where T denotes the total number of tokens in the long text. N refers to the number of chunks. t_c refers to the number of tokens per chunk.

In the default version of LongTest, we adopt the fixed-size chunking strategy for handling long text files in LongTest because it has been widely used in prior studies [24] and has demonstrated reliable performance in text processing tasks. Notably, this method is extremely fast (typically taking less than one second), making it well-suited to enhance the efficiency of LongTest.

Rationale: The reason we perform text splitting is that embedding models typically have input token limitations, meaning that they are constrained by the length of the input text and cannot process information beyond this limit. Therefore, if the long text l exceeds the token limit, the portion of the text exceeding this limit will be lost when converting the text into an embedding vector, resulting in information loss. Through chunk-based splitting, we aim to enhance the capture of information from the entire long text file.

Impact of Text Splitting on Semantic Coherence. The primary goal of introducing splitting strategies in our work is to enhance test prioritization effectiveness. While splitting may unavoidably introduce some degree of semantic fragmentation, without splitting, directly feeding long texts into the model would exceed input token limits, leading to truncation and more severe information loss. For example, in models such as BERT [37], which have a maximum input length of 512 tokens, any content beyond this limit would be truncated, resulting in severe information loss that is significantly greater than the semantic disruption potentially introduced by text splitting. As long as the downstream performance on test prioritization remains strong, these splitting strategies are considered practically valuable in this context. Additionally, to further reduce the potential impact of document splitting on semantic integrity, we compare the effectiveness of four different splitting strategies, including semantic-based, paragraph-based, structure-aware, and fixed-length splitting, which can be found in Section 5.7 (RQ7). The experimental results demonstrate that LongTest maintains its effectiveness across different splitting methods.

Furthermore, although recent large language models support substantially larger context windows, chunk-based text splitting remains practically important in our setting. This is mainly because: 1) Domain-specific small models trained on fine-grained datasets can typically achieve higher accuracy than large language models, particularly for classification tasks. This is because such models are specifically optimized for targeted domains and are trained on carefully curated task-specific data, allowing them to capture fine-grained semantic patterns and decision boundaries more effectively. In contrast, although large language models possess strong generalization capabilities, they are often not specifically optimized for specialized classification scenarios, which could result in relatively lower effectiveness in targeted domains compared to well-trained small models; 2) Large language models could lead to substantially higher computational and inference costs compared to small classical ML models, which could significantly increase the overall testing overhead. In contrast, well-trained classical ML models typically have much smaller model sizes, and can typically complete inference in less than one second. This makes them more suitable for efficient and scalable test prioritization, especially in resource-constrained environments.

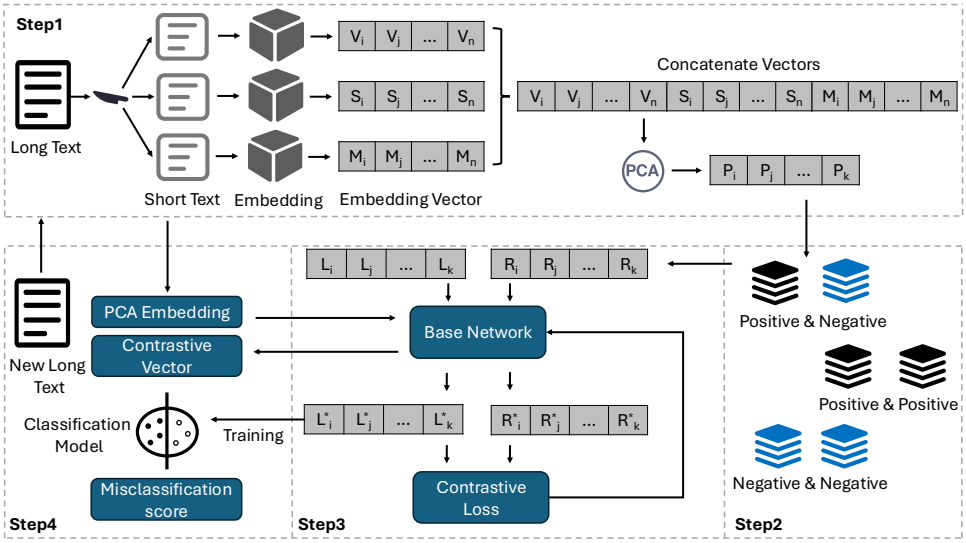


Fig. 1: Overview of LongTest

3.2.2 *Transforming Text into Embeddings*

Next, for each long text sample $l \in R$, we transform the obtained chunks $l = [c_1, c_2, \ldots, c_n]$ individually into embeddings using an embedding algorithm [38]. Then, we concatenate all the embeddings of the chunks of l to generate a final embedding vector V_l. This process is represented as Formula 3.

$$V_l = [\text{Embedding}(c_1), \text{Embedding}(c_2), \ldots, \text{Embedding}(c_n)] \quad (3)$$

where Embedding(c_i) denotes the embedding of the chunk c_i. V_l refers to the embedding vector of the long text sample l.

Here, we adopted MiniLM [38] as the embedding algorithm. MiniLM is a lightweight and efficient transformer-based language model that achieves a good balance between performance and computational cost.

Specifically, MiniLM employs a multi-objective distillation framework, in which the student model is trained to replicate the self-attention matrices and value-based similarity scores of a larger teacher model, rather than just its final output logits [39]. This strategy enables the student to learn both token-level dependencies and deeper structural semantics from the teacher's intermediate layers, leading to high-quality contextual embeddings.

In our system, MiniLM is applied to each chunk of a long text file. Each long text is split into fixed-size chunks, independently embedded using MiniLM, and the resulting chunk-level embeddings are concatenated to form a comprehensive vector representation. This allows the model to retain semantic information across the entire input. We selected MiniLM due to its strong empirical performance and lightweight nature.

To obtain the embedding of each chunk c_i, we adopt the following procedure. Specifically, we first clean the text to remove irrelevant characters or formatting. We then utilize the pretrained *all-MiniLM-L6-v2* model from the Sentence-Transformers library to compute sentence-level embeddings. Internally, the Sentence-Transformers framework tokenizes each input and passes it through the MiniLM model to obtain contextualized token representations. The final embedding vector is computed using mean pooling over all non-padding token embeddings, which is the default configuration of the framework and is widely adopted in practice for generating robust representations [40].

After splitting the long text into chunks and generating embeddings for each chunk, LongTest concatenates the embeddings of all chunks to construct the final representation of the long text. We adopt concatenation to combine the chunk embeddings because it preserves the semantic information of each chunk without introducing additional computational overhead. Compared with other aggregation methods, such as pooling-based approaches [40] and attention-based aggregation methods [41], concatenation provides a more effective strategy to preserve chunk-level semantic information while maintaining computational efficiency. Pooling-based methods typically compress chunk-level representations into a single vector and could limit the ability to capture fine-grained semantic information [41]. Attention-based aggregation methods require additional model parameters and increase computational complexity [42], which conflicts with the efficiency goal of LongTest. Therefore, we adopt simple concatenation, which allows LongTest to preserve chunk-level semantic information in the final representation while maintaining high computational efficiency.

Rationale: The reason we transform each long text sample $l \in R$ into an embedding vector is that text is unstructured data, which models cannot directly understand. Embeddings convert text into fixed-dimensional vector representations, preserving the semantic information of the text and facilitating subsequent analysis. Through embeddings, the text content of $l \in R$ can be used to train the ranking model in the following step (which is used to predict how likely a long text sample will be misclassified).

3.2.3 Dimensionality Reduction with PCA

In the previous step, for each long text sample l in the training set R, we obtained its embedding vector V_l. In this step, we perform dimensionality reduction on this embedding vector using the Principal Component Analysis (PCA) algorithm [17], aiming to reduce its dimensionality to decrease the overall runtime while preserving the essential semantic information of the long text. This process can be represented as Formula 4.

$$V_l^{PCA} = \mathrm{PCA}(V_l) \tag{4}$$

where V_l is the original embedding vector of each long text sample $l \in R$. PCA() denotes the Principal Component Analysis algorithm used for dimensionality reduction. V_l^{PCA} is the resulting lower-dimensional embedding of l.

Below, we provide the specific implementation details of the PCA algorithm.

- **Input Preparation**: For each long-text sample, we have obtained its embedding vector in the previous step (cf. Section 3.2.1 to Section 3.2.2). These embeddings are typically high-dimensional.
- **Mean Centering**: To standardize the data before dimensionality reduction, we subtract the mean vector (calculated from all training samples) from each individual embedding. This ensures that the data is centered around the origin.
- **Covariance Matrix Computation**: After mean-centering, we calculate the covariance matrix of the embeddings. This matrix captures how different dimensions of the embeddings vary with respect to each other.
- **Eigen Decomposition**: We perform eigenvalue decomposition on the covariance matrix and select the top components that capture the most variance in the data. The corresponding eigenvectors form a projection matrix that defines the new and lower-dimensional feature space.
- **Projection to Low-Dimensional Space**: Each high-dimensional embedding vector is then projected into a lower-dimensional space using the selected principal components. This transformation reduces the number of features while retaining as much of the original information as possible.

PCA was used in our implementation as the default dimensionality reduction technique. We made this choice because the PCA algorithms are very fast and have the ability to preserve the most significant variance in high-dimensional embedding vectors. Despite being a classical method, PCA remains prevalent in recent research because it has been demonstrated as a highly effective and stable method, particularly in scenarios where computational efficiency is crucial [43]. Nonetheless, LongTest is compatible with other dimensionality reduction techniques. Researchers are encouraged to integrate these techniques into LongTest depending on their domain requirements, resource constraints, or model preferences.

Rationale: Dimension reduction aims to reduce the overall runtime of LongTest and enhance the efficiency of test prioritization while preserving the features of the original data. High dimensionality can lead to the **curse of dimensionality**, which causes computational inefficiency and the risk of overfitting due to sparse data distribution in high-dimensional spaces.

Although some large language model (LLM) tools could potentially be used for feature generation in long texts, we did not adopt them in this work for the following three key reasons: **1) Computational Resource Cost** LLMs with long-context feature generation capabilities (e.g., GPT-4, Claude, LLaMA-2-70B with extended context) require extremely high GPU memory and computational power. In the setting of test input prioritization, where thousands of long documents must be processed, the cost of employing such LLMs becomes prohibitive. For example, in the EURLEX57K dataset, the test set contains 11,400 documents, and processing these tests would result in extremely high computational resource consumption. In contrast, embedding models (e.g., MiniLM, MPNet) are more lightweight and require comparatively little computational resources.

2) Time Efficiency Our task requires processing large-scale datasets (e.g., EURLEX57K with 57,000 documents). Using long-context LLMs would dramatically increase inference time, limiting the practicality of our approach. Our embedding-based method provides a balance between effectiveness and efficiency, making it feasible to deploy LongTest in real-world testing pipelines. 3) Importantly, compared to LLMs, our embedding approach is also based on the Transformer architecture. Thus, we still exploit its representational strength in a resource-efficient manner. Our experimental results demonstrate that LongTest consistently outperforms existing test prioritization methods, with improvements ranging from 9.61% to 16.31%, validating the effectiveness of our embedding-based design.

3.3 Step 2: Constructing Positive and Negative Pairs

In this step, we construct positive and negative pairs for training the contrastive learning model. The specific steps are as follows:

❶ First, for each long text sample $l \in R$, we use the text classification model M to be evaluated to predict the label of l, obtaining the prediction result $\hat{y}_l$.

❷ Given that the ground truth of l is y_l, we compare $\hat{y}_l$ with y_l to determine whether l is misclassified by the model. If l is misclassified, we label it as "positive" ($+$); otherwise, we label it as "negative" ($-$). This process is presented in Formula 5

$$z_l = \begin{cases} +1, & \text{if } \hat{y}_l \neq y_l \quad \text{(misclassified, labeled as "positive")} \\ -1, & \text{if } \hat{y}_l = y_l \quad \text{(correctly classified, labeled as "negative")} \end{cases} \tag{5}$$

where z_l represents whether sample l is misclassified by the model, with $z_l = +1$ indicating that the sample is misclassified (labeled as "positive") and $z_l = -1$ indicating that the sample is correctly classified (labeled as "negative"). Here, $\hat{y}_l$ is the predicted label generated by

the text classification model M, and y_l is the ground truth label of the sample l. If $\hat{y}_l \neq y_l$, it corresponds to misclassification. If $\hat{y}_l = y_l$, it corresponds to correct classification.

❸ After labeling all the samples in the training R as "positive" or "negative", we construct positive and negative sample pairs. Specifically, we construct three types of sample pairs in total: positive-negative, positive-positive, and negative-negative, which are presented as follows. These constructed pairs will be used in subsequent steps to train the contrastive learning model.
In the following, a and b represent two arbitrary data samples from the training dataset R.

- **Positive-Negative Pairs** ($\mathcal{P}_{PN}$) The positive-negative pair set $\mathcal{P}_{PN}$ includes pairs where $a, b \in R$, one sample has a positive label, and the other has a negative label:

$$\mathcal{P}_{PN} = \{(\mathbf{v}_a^{PCA}, \mathbf{v}_b^{PCA}) \mid a, b \in R, z_a = +1, z_b = -1, a \neq b\}.$$

- **Positive-Positive Pairs** ($\mathcal{P}_{PP}$) The positive-positive pair set $\mathcal{P}_{PP}$ includes pairs where $a, b \in R$ and both samples have positive labels:

$$\mathcal{P}_{PP} = \{(\mathbf{v}_a^{PCA}, \mathbf{v}_b^{PCA}) \mid a, b \in R, z_a = +1, z_b = +1, a \neq b\}.$$

- **Negative-Negative Pairs** ($\mathcal{P}_{NN}$) The negative-negative pair set $\mathcal{P}_{NN}$ includes pairs where $a, b \in R$ and both samples have negative labels:

$$\mathcal{P}_{NN} = \{(\mathbf{v}_a^{PCA}, \mathbf{v}_b^{PCA}) \mid a, b \in R, z_a = -1, z_b = -1, a \neq b\}.$$

where $\mathbf{v}_a^{PCA}$ and $\mathbf{v}_b^{PCA}$ refer to the PCA-reduced embedding vectors for samples a and b, respectively. z_a and z_b represent the labels (positive or negative) of samples a and b, respectively. $z_a = +1$ indicates that the sample a has a positive label, and $z_b = -1$ indicates that the sample b has a negative label. The condition $a \neq b$ ensures that a and b are not the same sample.

❹ Finally, we obtained the complete set of sample pairs:

$$\mathcal{P} = \mathcal{P}_{PN} \cup \mathcal{P}_{PP} \cup \mathcal{P}_{NN},$$

where $\mathcal{P}$ refers to the complete set of sample pairs, which is the union of $\mathcal{P}_{PN}$, $\mathcal{P}_{PP}$, and $\mathcal{P}_{NN}$. $\mathcal{P}_{PN}$ refers to the set of Positive-Negative Pairs. $\mathcal{P}_{PP}$ refers to the set of Positive-Positive Pairs. $\mathcal{P}_{NN}$ refers to the set of Negative-Negative Pairs.

Here, we adopt a controlled pair construction strategy instead of enumerating all possible combinations. Specifically, for each sample $l \in R$, we construct two pairs: one positive pair (paired with a misclassified sample) and one negative pair (paired with a correctly classified sample). Under this strategy, the total number of constructed pairs grows linearly with the dataset size rather than quadratically. This design significantly reduces the computational cost and memory overhead. Moreover, it ensures that each sample contributes equally to the training process, preventing the dominance of samples with large numbers of possible pair combinations.

3.4 Step 3: Training Contrastive Learning Model

To effectively distinguish between misclassified and correctly classified tests, we leverage a contrastive learning approach. Specifically, the base network of the contrastive learning model consists of three fully connected layers, each containing 128 units and employing sigmoid activation functions. Through this hierarchical structure, the model is capable of extracting abstract and meaningful feature representations from the test data.

We chose this architecture for two main reasons. First, the input to the contrastive model is the embedding vector after PCA-based dimensionality reduction, which still retains high-level semantic information extracted from the long-text tests. Therefore, the base network mainly serves as a transformation function that reorganizes the representation space to better separate misclassified and correctly classified tests, rather than learning complex representations from raw data. In this setting, a shallow fully connected network is sufficient. Second, a shallow fully connected network provides a balance between representation capacity and model simplicity. A deeper architecture may increase the risk of overfitting, while a simpler network could lack sufficient capacity to capture discriminative patterns. Moreover, this design follows common practices in metric learning and contrastive learning literature, where fully connected networks or multilayer perceptrons are widely used to transform embeddings into contrastive representation spaces [44], [45].

Using the pairs $\mathcal{P}$ generated in the previous step, we train the contrastive learning model. The contrastive loss function is designed to bring the same type pairs (i.e., Positive-Positive and Negative-Negative pairs) closer in the representation space, while pushing apart different-type pairs, such as Positive-Negative pairs. In this way, misclassified ("positive") tests are closer together in space, while misclassified ("positive") and correctly classified tests ("negative") are put farther apart. This could enhance the model's ability to differentiate between the two categories, supporting more effective test prioritization. Specifically, the contrastive loss function is presented as follows:

$$L = y \cdot D^2 + (1 - y) \cdot \max(0, m - D)^2 \tag{6}$$

where L denotes the overall contrastive loss value for a given input pair. $y \in \{0, 1\}$ is the label indicating whether the sample pair belongs to the same class (1) or not (0). m is a margin parameter (set to 3 in our experiments), which defines the minimum distance required between dissimilar pairs. D refers to the Euclidean distance between the embeddings of the two input samples, computed as $D = \|f(x_1) - f(x_2)\|_2$, where $f(\cdot)$ is the shared base network.

When $y = 1$ (i.e., similar pair): The loss term D^2 encourages the model to minimize the distance between embeddings, pulling similar samples closer together in the feature space. When $y = 0$ (i.e., dissimilar pair), the loss term $\max(0, m - D)^2$ applies a penalty only if the distance D is less than the margin m, pushing dissimilar samples further apart until their distance exceeds m.

After training, we obtain the contrastive learning model, which can be used to transform embedding vectors of a test input. Specifically, if we input the dimensionality-reduced embedding vector V_t^{PCA} of a long-text test t into the trained base network, a contrastive vector will be generated

based on it. This process is mathematically represented in Formula 7.

$$\mathbf{v}_t^{\text{cont}} = f_\theta\left(\mathbf{v}_t^{\text{PCA}}\right) \tag{7}$$

where $\mathbf{v}_t^{\text{PCA}}$ denotes the input vector, representing the dimensionality-reduced embedding of the long-text test t. f_θ denotes the base network, a transformation function parameterized by θ, which is learned during the training phase of the contrastive learning model. $\mathbf{v}_t^{\text{cont}}$ denotes the output vector, referred to as the "contrastive vector", which maps the long-text test in a space optimized for distinguishing between misclassified and correctly classified tests.

3.5 Step 4: Training Ranking Model for Prioritization

In the final step, we trained a Random Forest model [46] to classify between misclassified tests and correctly classified tests. We selected Random Forest as the ranking model in LongTest because it is faster compared to other ranking models, such as XGBoost [16] and LightGBM [47], while maintaining high effectiveness. Here, we select classical machine learning models for the following reasons. First, in LongTest, we leverage contrastive learning to learn feature representations for textual inputs. After this process, each textual input is transformed into a vector embedding, resulting in a tabular dataset representation. Prior studies [48], [49] have shown that, for classification tasks on tabular datasets, classical machine learning models can typically achieve higher performance compared to deep neural networks (DNNs). Second, compared to DNN-based approaches, classical machine learning models are generally more efficient, requiring less training time and fewer computational resources. Neural networks typically incur substantially higher computational overhead and longer training times [50]. Furthermore, Random Forests have demonstrated strong stability across diverse datasets, and they provide fast prediction speed with lower model complexity and maintenance requirements.

To train the ranking model in LongTest for test prioritization, we constructed a new training dataset, denoted as R_{RF}. This dataset was built based on the training dataset R from the original dataset. The construction process of R_{RF} is described as follows:

❶ **Extracting Contrastive Features:** Following the process described in Section 3.2 to Section 3.4, we first perform text preprocessing, dimensionality reduction, and contrastive learning on each long-text sample $l \in R$. We then transform each sample into a contrastive feature vector, denoted as $\mathbf{v}_l$.

❷ **Assigning Misclassification Labels:** For each sample $l \in R$, we compare the prediction result of the target model with the corresponding ground-truth label to determine whether the sample is misclassified. We assign a label of 1 to misclassified samples and a label of 0 to correctly classified samples.

❸ **Constructing the New Training Dataset:** We construct the new training dataset R_{RF} by using the contrastive feature vector $\mathbf{v}_l$ of each sample $l \in R$ as the input feature and the corresponding misclassification label as the target label. We then use the resulting dataset R_{RF} to train the Random Forest ranking model.

Test Input Ranking After training the initial ranking model, we obtain a trained standard binary Random Forest classifier. Based on this classifier, we further construct a modified variant that can directly produce a misclassification probability score for each test case. This variant is used as the final ranking model in LongTest to rank and prioritize test inputs according to their likelihood of being misclassified by the target model. Below, we describe the specific modifications in detail.

In the original Random Forest classifier, when a test sample is provided as input, the model classifies it as either misclassified or correctly classified (i.e., 0 or 1). In the modified variant, instead of returning the final binary prediction, we retrieve the intermediate probability score. This score represents the estimated probability that a given test sample belongs to class 1 (i.e., the probability that the sample is likely to be misclassified by the target model).

We refer to this probability value as the *misclassification score*, and use it to rank all test samples for prioritization. A higher misclassification score indicates a greater likelihood that the corresponding test sample will be misclassified by the target model. Therefore, all test samples are ranked in descending order according to their misclassification scores.

In this way, LongTest utilizes the estimated misclassification probability as the ranking score for each test sample, which is consistent with the idea of learning-based ranking approaches for test prioritization.

3.6 Usage of LongTest

In this section, we present the usage of LongTest. Given an unordered, unlabeled long-text test set $T = \{t_1, t_2, \ldots, t_n\}$ and a text classification model M to be evaluated, LongTest outputs a ranked test set. The process consists of the following steps: 1) test sample preprocessing, 2) contrastive embedding generation, 3) misclassification probability estimation, and 4) test prioritization. Moreover, we demonstrate the training process of the contrastive learning model and the ranking model of LongTest.

Workflow of LongTest In the following, we present how LongTest works on an unordered, unlabeled long-text test set.

❶ For each long-text test sample t_i in T, LongTest processes it using the text preprocessing and dimensionality reduction steps described in Step 1 (cf. Section 3.2), obtaining the reduced-dimensional embedding for each test. The resulting set of embeddings is represented as:

$$\mathcal{T}_{\text{emb}} = \left\{\mathbf{v}_{t_1}^{\text{PCA}}, \mathbf{v}_{t_2}^{\text{PCA}}, \ldots, \mathbf{v}_{t_n}^{\text{PCA}}\right\}$$

where $\mathbf{v}_{t_i}^{\text{PCA}}$ represents the PCA-reduced embedding vector for the test sample t_i. $\mathcal{T}_{\text{emb}}$ represents the set of PCA-reduced embedding vectors for all test samples $t_1, t_2, \ldots, t_n$.

❷ Next, based on the contrastive learning model trained in Step 3 (cf. Section 3.4), LongTest inputs the reduced-dimensional vector of each test t_i into the contrastive learning model to obtain its contrastive embedding vector. This is represented as:

$$\mathcal{T}_{\text{cont}} = \left\{\mathbf{v}_{t_1}^{\text{cont}}, \mathbf{v}_{t_2}^{\text{cont}}, \ldots, \mathbf{v}_{t_n}^{\text{cont}}\right\}$$

where $\mathbf{v}_{t_i}^{\text{cont}}$ denotes the contrastive embedding vector for each test sample t_i. $\mathcal{T}_{\text{cont}}$ represents the set of contrastive embedding vectors for all test samples $t_1, t_2, \ldots, t_n$.

❸ Subsequently, based on each sample's contrastive learning vector $\mathbf{v}_{t_i}^{\text{cont}}$, LongTest utilizes the Random Forest model trained in Step 4 (cf. Section 3.5) to estimate the probability of each test sample t_i being misclassified by the model M. This is represented as:

$$\mathcal{P}_{\text{mis}} = \{P_{\text{mis}}(t_1), P_{\text{mis}}(t_2), \ldots, P_{\text{mis}}(t_n)\}$$

where $P_{\text{mis}}(t_i)$ represents the probability of test sample t_i being misclassified by the model M. $\mathcal{P}_{\text{mis}}$, represents the set of misclassification probabilities for all test samples $t_1, t_2, \ldots, t_n$.

❹ Finally, LongTest ranks all the tests in the test set T based on their probabilities to be misclassified by the model, resulting in a sorted test set $\mathcal{T}_{\text{ranked}}$:

$$\mathcal{T}_{\text{ranked}} = \{t_{(1)}, t_{(2)}, \ldots, t_{(n)}\}$$

such that:

$$P_{\text{mis}}(t_{(1)}) \geq P_{\text{mis}}(t_{(2)}) \geq \cdots \geq P_{\text{mis}}(t_{(n)})$$

where $P_{\text{mis}}(t_{(i)})$ denotes the probability that the sample $t_{(i)}$ will be misclassified by the model M. The subscript indices $(1), (2), \ldots, (n)$ represent the ranked order of the test samples after sorting by their misclassification probabilities. For example, $t_{(1)}$ refers to the test sample with the highest misclassification probability.

Training and Testing of LongTest In the following, we present the training and testing process of LongTest. We explain the source of training data for LongTest's contrastive learning models and ranking model, as well as the testing data used to evaluate LongTest and the compared test prioritization approaches.

1) First, we split the original dataset for LongTest into a training set (denoted as R) and a test set (denoted as T). Through the steps in Section 3.3, we construct positive and negative pairs to train the contrastive learning model.
2) Once the contrastive learning model is trained, we input the original training set R to the well-trained contrastive learning model, transforming each sample into a contrastive sample. After transformation, the misclassified samples become closer to each other in the feature space.
3) From the previous step, we obtained a new feature set (feature set based on contrastive learning). Next, we construct a new label set. Based on the new feature set and the new label set, we obtain a new training set, denoted as R', which is used to train LongTest's ranking model. The method for constructing the new label set is as follows: for a sample, if the sample is misclassified by the long text classification model, it is labeled as 1. Otherwise, it is labeled as 0.
4) We used the new training set R' to train LongTest's ranking model.
5) Throughout the entire process, the test set T remains untouched and is used to evaluate the effectiveness of LongTest and other test prioritization methods.

4 Study Design

In this section, we comprehensively present our study design from various perspectives. Specifically, Section 4.1 presents the research questions that served as the guiding framework for our investigation. Section 4.2 presents the datasets and models employed to assess the effectiveness of LongTest. Section 4.3 presents the measurements we used for evaluation. Section 4.4 presents the test prioritization approaches we utilized to compare with LongTest. Section 4.5 describes the implementation and configuration setup utilized in our study.

4.1 Research Questions

Our experimental evaluation addresses the following research questions.

- **RQ1: How does LongTest perform in prioritizing tests for long text files?**
  We evaluate the performance of LongTest by comparing it with several existing test prioritization approaches [10], [9]. We conduct statistical analysis [51], [52] to prove that the observed improvements achieved by LongTest over the existing approaches are statistically significant.
- **RQ2: How does the number of chunks affect LongTest effectiveness?**
  To perform test prioritization, LongTest first divides a long text file into smaller chunks, converts these chunks into embeddings, and then concatenates them. In this research question, we investigate how splitting into different numbers of chunks impacts the effectiveness of LongTest.
- **RQ3: What is the impact of different embedding models on LongTest?**
  In the test prioritization process of LongTest, chunks of text are converted into vectors using an embedding model. This research question investigates the impact of different embedding models on the effectiveness of LongTest.
- **RQ4: What is the impact of reducing to different dimensions on LongTest?**
  Within the LongTest framework, we apply Principal Component Analysis (PCA) [17] to reduce the dimensionality of text embedding vectors. This research question investigates the impact of reducing text vectors to different dimensions on the effectiveness of LongTest.
- **RQ5: How do the main parameters influence the effectiveness of LongTest?**
  We investigate how different parameter settings of LongTest affect its effectiveness and aim to identify parameter configurations that provide relatively higher effectiveness.
- **RQ6: Does each component contribute to LongTest?**
  LongTest comprises core components, including contrastive learning and PCA for dimensionality reduction. This research question investigates the contributions of these components to the effectiveness of LongTest through an ablation study.
- **RQ7: What is the impact of different splitting strategies on LongTest?**
  In the default setting, LongTest adopts a fixed-size chunking strategy to divide long text files. In this research ques-

tion, we investigate the impact of using different chunk splitting methods (e.g., semantic-based, paragraph-based, or structure-aware segmentation) on the effectiveness of LongTest.

- **RQ8: How do different loss functions used in contrastive learning impact LongTest?**
  In the original design, LongTest employs a Euclidean-distance-based contrastive loss to guide contrastive learning. In this research question, we explore the impact of different loss functions on the effectiveness of LongTest. We compare a set of widely adopted loss functions in contrastive learning, including SimSCE loss [45], contrastive loss with cosine similarity [53], and Manhattan distance-based loss [54].
- **RQ9: Can LongTest guide the retraining of long text classification models?**
  In real-world scenarios, long-text classification models can be affected by distribution shifts, noisy samples, and even adversarial inputs, which may lead to performance reduction. Fully re-labeling data or retraining from scratch is usually costly and inefficient. In this research question, we investigate whether LongTest can effectively guide retraining by selecting more informative samples for retraining and efficiently improve model accuracy.
- **RQ10: What is the impact of different ranking models on the effectiveness of LongTest?**
  In LongTest, a ranking model is used to estimate the misclassification probability of test inputs and prioritize them accordingly. The default model we adopted is Random Forest [46]. In this research question, we investigate how different ranking models influence the effectiveness of LongTest and aim to identify the ranking model that is most suitable for LongTest to perform test prioritization.
- **RQ11: What is the impact of document length on the effectiveness of LongTest?**
  We investigate whether the effectiveness of LongTest remains consistent across long-text and short-text documents. We aim to examine whether LongTest can consistently outperform existing test prioritization methods under different document length settings.
- **RQ12: What is the impact of target model accuracy on LongTest?**
  In practical scenarios, text classification models could achieve different levels of predictive accuracy due to differences in model architectures, training datasets, or training configurations. These variations may lead to different distributions of misclassified samples, which could potentially affect the performance of test prioritization methods. This research question aims to investigate whether the effectiveness of LongTest is influenced by different predictive accuracy levels of the target model.

4.2 Models and Datasets

To evaluate the effectiveness of LongTest and the compared test prioritization approaches, we use a set of 45 subjects, including 3 long-text datasets and 15 text classification models. The essential details and the matching relationship between the long-text datasets and the models are presented in Table 1.

TABLE 1: Datasets and Models

ID	Dataset	Ratio of Long Docs	# Size	Models
1	CancerDoc	98.7%	7569	DistilRoBERTa-DNN
2	CancerDoc	98.7%	7569	DistilRoBERTa-DT
3	CancerDoc	98.7%	7569	DistilRoBERTa-LR
4	CancerDoc	98.7%	7569	DistilRoBERTa-RF
5	CancerDoc	98.7%	7569	DistilRoBERTa-TabNet
6	CancerDoc	98.7%	7569	MPNet-DNN
7	CancerDoc	98.7%	7569	MPNet-DT
8	CancerDoc	98.7%	7569	MPNet-LR
9	CancerDoc	98.7%	7569	MPNet-RF
10	CancerDoc	98.7%	7569	MPNet-TabNet
11	CancerDoc	98.7%	7569	MiniLM-DNN
12	CancerDoc	98.7%	7569	MiniLM-DT
13	CancerDoc	98.7%	7569	MiniLM-LR
14	CancerDoc	98.7%	7569	MiniLM-RF
15	CancerDoc	98.7%	7569	MiniLM-TabNet
16	EURLEX57K	51.3%	57000	DistilRoBERTa-DNN
17	EURLEX57K	51.3%	57000	DistilRoBERTa-DT
18	EURLEX57K	51.3%	57000	DistilRoBERTa-LR
19	EURLEX57K	51.3%	57000	DistilRoBERTa-RF
20	EURLEX57K	51.3%	57000	DistilRoBERTa-TabNet
21	EURLEX57K	51.3%	57000	MPNet-DNN
22	EURLEX57K	51.3%	57000	MPNet-DT
23	EURLEX57K	51.3%	57000	MPNet-LR
24	EURLEX57K	51.3%	57000	MPNet-RF
25	EURLEX57K	51.3%	57000	MPNet-TabNet
26	EURLEX57K	51.3%	57000	MiniLM-DNN
27	EURLEX57K	51.3%	57000	MiniLM-DT
28	EURLEX57K	51.3%	57000	MiniLM-LR
29	EURLEX57K	51.3%	57000	MiniLM-RF
30	EURLEX57K	51.3%	57000	MiniLM-TabNet
31	FakeNews	32.4%	44898	DistilRoBERTa-DNN
32	FakeNews	32.4%	44898	DistilRoBERTa-DT
33	FakeNews	32.4%	44898	DistilRoBERTa-LR
34	FakeNews	32.4%	44898	DistilRoBERTa-RF
35	FakeNews	32.4%	44898	DistilRoBERTa-TabNet
36	FakeNews	32.4%	44898	MPNet-DNN
37	FakeNews	32.4%	44898	MPNet-DT
38	FakeNews	32.4%	44898	MPNet-LR
39	FakeNews	32.4%	44898	MPNet-RF
40	FakeNews	32.4%	44898	MPNet-TabNet
41	FakeNews	32.4%	44898	MiniLM-DNN
42	FakeNews	32.4%	44898	MiniLM-DT
43	FakeNews	32.4%	44898	MiniLM-LR
44	FakeNews	32.4%	44898	MiniLM-RF
45	FakeNews	32.4%	44898	MiniLM-TabNet

4.2.1 Datasets

In our research, we utilized three long text datasets: EURLEX57K [55], FakeNews [56], and Cancer Text Documents [7]. The reasons for selecting these datasets to evaluate LongTest are as follows: these datasets have been extensively studied in academia and are widely used and discussed by researchers and developers on data science competition platforms (such as Kaggle). Additionally, these datasets contain a considerable proportion of long texts. For example, the CancerDoc dataset comprises 98.7% long texts, while the EURLEX57K and FakeNews datasets include 51.3% and 32.4% long texts, respectively. Therefore, these datasets are suitable for testing the long text-oriented test prioritization approach LongTest. Here, based on prior research, texts with more than 512 tokens are regarded as long texts [57].

- **EURLEX57K** [55] The EURLEX57K dataset is a large-scale, publicly available dataset consisting of 57k English-language EU legislative documents from the EUR-LEX portal.
- **FakeNews** [56] The FakeNews dataset is a publicly available dataset containing news articles with titles and full-text content. This dataset is designed for text classification

tasks aimed at distinguishing between real and fake news. In this dataset, the proportion of long texts is 32.4%. According to an existing study on long-text classification [24], this proportion falls within a reasonable range, and therefore we used this dataset in our experiments.
- **CancerDoc** [7] The Cancer Text Documents dataset is a publicly available dataset consisting of cancer-related research articles, including abstracts and full papers. This dataset focuses on lengthy research papers with more than six pages. It is designed for text classification tasks aimed at analyzing and categorizing cancer-focused content. Examples of labels include "Thyroid_Cancer" and "Colon_Cancer".

4.2.2 Models

To evaluate the effectiveness of LongTest, we employed 15 well-established text classification models, specifically: DistilRoBERTa-DT [58], [59], DistilRoBERTa-DT [58], [60], DistilRoBERTa-LR [58], [61], DistilRoBERTa-RF [58], [62], DistilRoBERTa-TabNet [58], [63], MPNet-DNN [64], [59], MPNet-DT [64], [60], MPNet-LR [64], [61], MPNet-RF [64], [62], MPNet-TabNet [64], [63], MiniLM-DNN [38], [59], MiniLM-DT [38], [60], MiniLM-LR [38], [61], MiniLM-RF [38], [62], and MiniLM-TabNet [38], [63]. These combo models follow a widely adopted pipeline that combines pretrained embedding models with downstream classifiers, which has been extensively used in prior studies on text classification [40]. In this paradigm, embedding models are employed to capture semantic representations of text, while ranking model (e.g., Random Forest [46], Logistic Regression [61]) are used for downstream prediction tasks. Moreover, the individual components used in our study have been extensively validated in prior research and are widely used in text classification tasks [40], [65], [66], [64], [67], [68], [69].

Although our evaluation was conducted using these 15 models, the proposed LongTest framework is not limited to these specific models. LongTest is primarily designed to address long-text scenarios and can be applied to any text classification model. This is explained in detail in Section 6.1.

4.3 Measurements

To evaluate the effectiveness of LongTest and the comparative test prioritization methods, following prior work [9], we used two classic test prioritization metrics: Average Percentage of Fault-Detection (APFD) and Percentage of Fault Detected (PFD). Below, we provide a detailed explanation of these two metrics.

4.3.1 Average Percentage of Fault-Detection (APFD)

APFD [70] is a widely used metric for evaluating the effectiveness of test prioritization techniques. Higher APFD values indicate faster rates of detecting misclassified test inputs. The APFD value can be computed using Formula 8.

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{M} o_i}{M \cdot N} + \frac{1}{2N} \tag{8}$$

where N refers to the total number of test inputs. M denotes the number of misclassified test cases by the model, and o_i refers to the position of the i-th misclassified test case in the test set after prioritization. Following the existing study [9], we normalize the APFD value to $[0, 1]$. If the APFD value of a test prioritization approach is closer to 1, we consider this prioritization method more effective. In the following, we explain the corresponding reasons:

First, the larger the APFD value of a test prioritization method, the smaller the value of $\sum_{i=1}^{M} o_i$ in Formula 8, since both M and N are constants. Here, $\sum_{i=1}^{M} o_i$ represents the sum of indices of all misclassified tests within the prioritized test set. A smaller value of this sum indicates that more misclassified tests have been prioritized towards the front, suggesting that the test prioritization method is more effective at detecting misclassifications. Therefore, a test prioritization approach with high APFD is considered more effective.

4.3.2 Percentage of Fault Detected (PFD)

PFD calculates the percentage of misclassified tests detected out of all misclassified tests when prioritizing a certain percentage of the data. If a test prioritization method has high PFD values, it means the method can detect more misclassifications and is, therefore, more effective. PFD is calculated based on Formula 9. In our experiment, we calculated the PFD values for each test prioritization method when prioritizing 10%, 20%, 30%, 40%, and 50% of the tests, represented as PFD-10, PFD-20, PFD-30, PFD-40, and PFD-50, respectively.

$$\text{PFD} = \frac{|\mathcal{N}_d|}{|\mathcal{N}|} \tag{9}$$

where $\mathcal{N}_d$ denotes the set of misclassified tests that have been detected. $\mathcal{N}$ represents the total set of misclassified tests.

4.3.3 Normalized Average Percentage of Fault Detection (NAPFD)

In addition to APFD and PFD, we also adopt the Normalized Average Percentage of Fault Detection (NAPFD) metric [71] to further evaluate the effectiveness of test prioritization techniques in detecting faults. The NAPFD value can be computed using the following Formula 10:

$$NAPFD = p - \frac{TF_1 + TF_2 + \cdots + TF_m}{n \cdot m} + \frac{p}{2n} \tag{10}$$

where p denotes the ratio of unique faults detected by the prioritized test set to the total number of faults in the system. m represents the total number of faults. n is the total number of test inputs. TF_i is the index of the test case in the prioritized sequence T' that detects fault f_i.

NAPFD directly evaluates the effectiveness of fault-revealing test cases across all faults in the system. Thus, a high NAPFD score indicates that the prioritization method not only detects more faults but also does so earlier, which is especially important in cost-sensitive testing scenarios.

4.4 Compared Approaches

To evaluate the effectiveness of LongTest, we adopted five test prioritization methods as comparison approaches, including four confidence-based methods and a baseline method (random selection). These prioritization methods were selected for two main reasons: first, all methods can be applied to prioritizing long text files, and their implementations are open-source. Second, the effectiveness of these methods has been demonstrated [10], [9]. Below, we provide a detailed description of the principles and workflows of these prioritization methods.

- **DeepGini** [9] DeepGini is a confidence-based test prioritization approach, which calculates the Gini score of the test to estimate how likely this test will be misclassified. If the score is higher, it means that this test is more likely to be misclassified. The calculation process of the Gini score is presented in Formula 11.

$$\text{Gini}(t) = 1 - \sum_{i=1}^{N} \left(p_i(t)\right)^2 \tag{11}$$

  where $p_i(t)$ represents the probability that the input t to be classified to category i. N denotes the total number of possible categories.
- **Vanilla Softmax** [10] The Vanilla Softmax (VanillaSM) approach quantifies the uncertainty of the model's prediction for a given test by measuring the difference between the model's most confident prediction and the ideal value of 1. The greater this difference, the less confident the model is in its prediction for this test, indicating that the test is more likely to be misclassified. This difference value is calculated by Formula 12.

$$\text{VanillaSM}(t) = 1 - \max_{i=1}^{N} p_i(t) \tag{12}$$

  where $p_i(t)$ refers to the probability that the input t is classified into category i. Therefore, $\max_{i=1}^{N} p_i(t)$ denotes the model's highest confidence prediction for the test input t.
- **Prediction-Confidence Score** The Prediction-Confidence Score (PCS) quantifies the model's confidence in its prediction for a given test input by calculating the difference between the probability of the predicted class and that of the second most confident class in the softmax output. If a test's PCS value is small, it indicates that the model is less confident on this test, and this test is more likely to be misclassified. The computation of PCS is presented in Formula 13.

$$\text{PCS}(t) = p_{\max}(t) - p_{\text{second}}(t) \tag{13}$$

  where $p_{\max}(t)$ denotes the probability associated with the model's most confident prediction for the test input t, and $p_{\text{second}}(t)$ represents the probability associated with the model's second most confident prediction for the test input t.
- **Entropy** [10] Entropy measures the model's confidence in a test input by calculating the entropy of the softmax likelihood distribution. If the entropy value for a test input is higher, it indicates that the model is less confident in its prediction, and this test input is more likely to be misclassified.
- **DATIS** [72] DATIS is a distance-aware test input selection method for DNNs. First, it selects test inputs using improved uncertainty scores based on the distances between test samples and their nearest neighbors in the training set. Then, it filters out redundant inputs by analyzing distances among the selected samples to maximize fault coverage and testing efficiency. To the best of our knowledge, DATIS remains one of the most representative state-of-the-art methods for text-based test prioritization. This makes it a strong baseline for direct comparison with our approach.
- **Dissector** [73] Dissector is an input validation approach that distinguishes beyond-inputs from within-inputs by analyzing the prediction process across layers of a DNN. First, it constructs a sequence of sub-models from a trained model, each representing partial knowledge up to a certain layer. Then, for a given test input, it collects prediction snapshots from these sub-models to form a prediction profile. Finally, it evaluates the input's validity by measuring whether the model's confidence in the final prediction increases consistently across layers, thereby producing a normalized score to indicate the likelihood that the input lies within the model's handling capability.
- **SelfChecker** [74] SelfChecker is a self-checking approach that assesses the reliability of DNN predictions using internal layer features. It models the feature distributions of each layer and class via kernel density estimation (KDE) based on training data. Then, for a given test input, it infers layer-wise class predictions by comparing the input's features to these learned distributions. Next, it selects an optimal subset of layers and evaluates the consistency between the inferred classes and the model's final prediction to determine whether to trigger an alarm.
- **Random selection** [75] Random selection determines the execution order of test inputs in a fully randomized manner.

4.5 Implementation and Configuration

We implemented LongTest in Python 3.7.2, utilizing TensorFlow 2.2.0 [76], PyTorch 1.11.0 [77], sklearn 0.24.2 [78], and SentenceTransformer 2.2.2 [79]. For the approaches used for comparison, we employed the available implementations provided by their respective authors [9], [10]. The accuracy of each model on each subject is shown in Table 2. For example, the DistilRoBERTa-DNN model achieves an accuracy of 0.721 on the CancerDoc dataset, which indicates that approximately 72.1% of the samples are correctly classified and 27.9% are misclassified. Based on existing studies [80], [81], this proportion of misclassified samples falls within a reasonable range and does not lead to a severe class imbalance issue. Our experimental setup involved conducting experiments on NVIDIA Tesla V100 32GB GPUs. For the data analysis, we utilized a MacBook Pro laptop running Mac OS Sonoma 14.3, equipped with an Intel Core i9 CPU and 64 GB of RAM.

TABLE 2: Accuracy of Classification Models Across Datasets

Models	Dataset		
	CancerDoc	EURLEX57K	FakeNews
DistilRoBERTa-DNN	0.721	0.848	0.853
DistilRoBERTa-DT	0.861	0.850	0.771
DistilRoBERTa-LR	0.924	0.794	0.819
DistilRoBERTa-RF	0.825	0.851	0.779
DistilRoBERTa-TabNet	0.786	0.867	0.725
MPNet-DNN	0.731	0.855	0.875
MPNet-DT	0.751	0.857	0.763
MPNet-LR	0.758	0.824	0.842
MPNet-RF	0.779	0.858	0.780
MPNet-TabNet	0.819	0.785	0.761
MiniLM-DNN	0.731	0.876	0.841
MiniLM-DT	0.714	0.850	0.712
MiniLM-LR	0.789	0.864	0.822
MiniLM-RF	0.741	0.841	0.817
MiniLM-TabNet	0.729	0.838	0.789

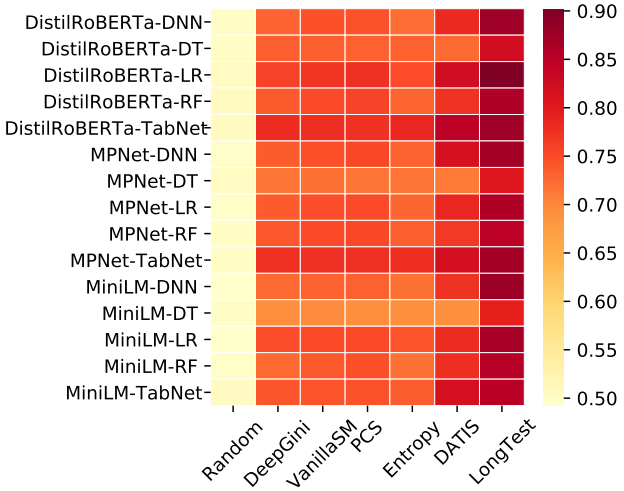


Fig. 2: Heatmap of APFD values for different test prioritization approaches across models

5 Results and Analysis

5.1 RQ1: Performance of LongTest

Objectives: We aim to evaluate the effectiveness of LongTest, a novel approach for prioritizing long text files, by comparing its performance with existing test prioritization techniques.

Experimental design: To comprehensively evaluate the performance of LongTest, our experimental design focused on three key aspects: effectiveness evaluation, statistical analysis, and efficiency evaluation.

- **Effectiveness evaluation** To assess the effectiveness of LongTest, we conducted experiments on a total of 45 subjects. Each subject represents a unique model-dataset pair. For instance, the subject (CancerDoc, MiniLM-DT) refers to the case where we use the dataset CancerDoc with the model MiniLM-DT to evaluate the test prioritization approaches. Table 1 provides basic information of all the subjects. We selected a set of well-established test prioritization methods (cf. Section 4.4) as the comparison approaches and adopted three classical evaluation metrics, namely APFD, PFD and NAPFD, for evaluation. Detailed explanations of these metrics can be found in Section 4.3.
- **Statistical analysis** Due to the inherent randomness in the model training process, we conducted statistical analysis by running all the experiments ten times and reported the average results. Moreover, we calculated the p-value and effect size to validate the stability and statistical significance of the experimental results.
  - **P-value:** To calculate the p-value, we employed the **paired two-sample t-test** [51], a widely used statistical method for evaluating differences between two data sets. Based on prior work [82], we consider differences between two data sets to be statistically significant if the p-value is less than 10^{-5}.
  - **Effect size:** We quantified the magnitude of the difference between the two sets of results using effect size, specifically employing Cohen's d [52] as a measure. Here, $|d| < 0.2$ implies a "negligible" effect, $|d| < 0.5$ implies a "small" effect, $|d| < 0.8$ implies a "medium" effect, and values above 0.8 imply a "large" effect. To ensure that the difference between the results of LongTest and the compared approach is "non-negligible", we require that the value of $d \geq 0.2$.
- **Efficiency evaluation** We evaluated the efficiency of LongTest and the compared test prioritization approaches by quantifying their runtime. Our goal is to gain insights into the computational efficiency and potential for practical application of LongTest.

Results: The experimental results of RQ1 is presented in Table 3, Table 4, Table 5, Table 6, Table 7, Table 8, Table 9 and Table 10. These tables respectively show the effectiveness evaluation using APFD, the statistical analysis, the effectiveness evaluation using PFD, and the efficiency evaluation.

Effectiveness comparison (in terms of APFD). Table 3 compares the effectiveness of LongTest and other approaches across different datasets and models based on the APFD metric. The comparison methods include: DATIS, DeepGini, VanillaSM, PCS, Entropy, and Random. We use gray highlighting to indicate the best-performing test prioritization method in each case. We see that LongTest outperforms the compared approaches in 93.3% (42 out of 45) of the cases. Specifically, on the CancerDoc dataset, LongTest achieves the best effectiveness in all cases. On the FakeNews dataset, LongTest performs best in 14 out of 15 cases. Table 4 shows an overall comparison between LongTest and other test prioritization approaches, including the number of best-performing cases for each method, the average APFD for each approach, and the average improvement of LongTest over the other test prioritization methods. The average effectiveness of LongTest is 0.856, whereas the average APFD values of the other methods range from 0.501 to 0.781. The improvement of LongTest over the other methods ranges from 9.61% to 70.86%. Based on the experimental results, we conclude that LongTest outperforms the compared test prioritization approaches in terms of the APFD metric in the vast majority of cases.

Figure 2 provides a heatmap visualization of the APFD results across different models, where each value represents the average APFD over the three datasets for the corresponding model. In the figure, darker colors indicate higher APFD values. The heatmap clearly shows that LongTest consistently achieves the highest APFD values across all

TABLE 3: Effectiveness comparison among LongTest, DeepGini, VanillaSM, PCS, Entropy, DATIS, and random selection in terms of the APFD values

Data	Model	Approach						
		Random	DeepGini	VanillaSM	PCS	Entropy	DATIS	**LongTest**
CancerDoc	DistilRoBERTa-DNN	0.511	0.650	0.693	0.680	0.647	0.759	0.861
	DistilRoBERTa-DT	0.518	0.753	0.754	0.753	0.753	0.729	0.930
	DistilRoBERTa-LR	0.511	0.828	0.837	0.845	0.814	0.919	0.980
	DistilRoBERTa-RF	0.525	0.734	0.760	0.769	0.723	0.801	0.912
	DistilRoBERTa-TabNet	0.495	0.735	0.734	0.733	0.738	0.792	0.893
	MPNet-DNN	0.512	0.638	0.662	0.676	0.620	0.791	0.865
	MPNet-DT	0.515	0.689	0.693	0.689	0.691	0.688	0.875
	MPNet-LR	0.488	0.695	0.701	0.702	0.685	0.741	0.879
	MPNet-RF	0.515	0.704	0.725	0.732	0.693	0.749	0.889
	MPNet-TabNet	0.511	0.740	0.741	0.741	0.740	0.778	0.908
	MiniLM-DNN	0.492	0.641	0.646	0.646	0.630	0.667	0.865
	MiniLM-DT	0.506	0.702	0.704	0.704	0.701	0.698	0.858
	MiniLM-LR	0.490	0.686	0.686	0.683	0.682	0.714	0.894
	MiniLM-RF	0.487	0.678	0.701	0.710	0.673	0.730	0.870
	MiniLM-TabNet	0.522	0.704	0.703	0.702	0.705	0.791	0.865
EURLEX57K	DistilRoBERTa-DNN	0.494	0.806	0.806	0.806	0.806	0.808	0.900
	DistilRoBERTa-DT	0.505	0.755	0.755	0.754	0.754	0.755	0.808
	DistilRoBERTa-LR	0.500	0.756	0.755	0.755	0.755	0.797	0.883
	DistilRoBERTa-RF	0.497	0.797	0.799	0.798	0.793	0.802	0.881
	DistilRoBERTa-TabNet	0.522	0.835	0.836	0.837	0.835	0.912	0.893
	MPNet-DNN	0.488	0.829	0.823	0.819	0.838	0.845	0.908
	MPNet-DT	0.497	0.767	0.768	0.768	0.766	0.757	0.825
	MPNet-LR	0.503	0.789	0.790	0.790	0.785	0.827	0.895
	MPNet-RF	0.491	0.821	0.821	0.818	0.817	0.822	0.877
	MPNet-TabNet	0.499	0.817	0.817	0.817	0.818	0.880	0.868
	MiniLM-DNN	0.491	0.845	0.843	0.842	0.846	0.866	0.918
	MiniLM-DT	0.496	0.743	0.743	0.743	0.741	0.740	0.814
	MiniLM-LR	0.493	0.836	0.834	0.834	0.838	0.855	0.911
	MiniLM-RF	0.488	0.796	0.796	0.795	0.791	0.823	0.880
	MiniLM-TabNet	0.493	0.776	0.780	0.783	0.761	0.892	0.895
FakeNews	DistilRoBERTa-DNN	0.497	0.737	0.746	0.753	0.720	0.781	0.868
	DistilRoBERTa-DT	0.484	0.697	0.697	0.695	0.695	0.692	0.734
	DistilRoBERTa-LR	0.504	0.697	0.728	0.737	0.684	0.760	0.844
	DistilRoBERTa-RF	0.501	0.686	0.700	0.708	0.676	0.723	0.790
	DistilRoBERTa-TabNet	0.504	0.776	0.768	0.758	0.788	0.842	0.836
	MPNet-DNN	0.496	0.747	0.759	0.769	0.735	0.813	0.842
	MPNet-DT	0.499	0.693	0.694	0.691	0.693	0.695	0.720
	MPNet-LR	0.497	0.730	0.759	0.764	0.717	0.795	0.809
	MPNet-RF	0.502	0.700	0.711	0.714	0.693	0.734	0.783
	MPNet-TabNet	0.498	0.780	0.779	0.778	0.781	0.802	0.835
	MiniLM-DNN	0.495	0.689	0.705	0.711	0.680	0.791	0.849
	MiniLM-DT	0.499	0.645	0.645	0.644	0.644	0.646	0.710
	MiniLM-LR	0.494	0.727	0.738	0.741	0.716	0.774	0.804
	MiniLM-RF	0.512	0.704	0.723	0.737	0.692	0.787	0.817
	MiniLM-TabNet	0.503	0.750	0.752	0.754	0.746	0.773	0.794

TABLE 4: Effectiveness improvement of LongTest over the compared approaches in terms of the APFD values

Approach	# Best cases	Average APFD	Improvement(%)
Random	0	0.501	70.86
DeepGini	0	0.741	15.52
VanillaSM	0	0.747	14.59
PCS	0	0.749	14.28
Entropy	0	0.736	16.31
DATIS	3	0.781	9.61
LongTest	42	0.856	-

the model configurations, which is reflected by the darker color intensity in the LongTest column compared with other approaches. This visual pattern further confirms the high effectiveness of LongTest over existing methods in terms of test prioritization effectiveness.

We further compare LongTest with input validation approaches, Dissector [73] and SelfChecker [74]. Since both Dissector and SelfChecker rely on layer-wise outputs of neural networks, they can only be applied to DNN models and are not applicable to classical machine learning models (e.g., decision trees and random forests), which lack such layered structures. Therefore, we conduct the comparison between these methods and LongTest on DNN models. Table 8 presents the comparison results. In the table, gray shading highlights the best-performing method in each case. As shown in Table 8, LongTest consistently outperforms both Dissector and SelfChecker across all datasets. For example, on the CancerDoc dataset, LongTest achieves a score of 0.876, compared to 0.694 for Dissector and 0.664 for SelfChecker. Similar improvements are also observed on EURLEX57K (0.869 vs. 0.823 and 0.751) and FakeNews (0.837 vs. 0.756 and 0.727). These results further demonstrate that LongTest consistently outperforms the compared methods, Dissector and SelfChecker.

Moreover, we applied the trained LongTest models to the BBCNews Dataset [83], which belongs to a domain similar to that of the original dataset used for training the ranking models. We aimed to evaluate the generalizability and cross-domain applicability of LongTest on similar-domain data. Table 10 presents the effectiveness comparison between LongTest and the compared approaches in terms of APFD. As shown in the table, LongTest achieves the highest APFD value of 0.591, outperforming all the compared approaches, including Entropy (0.568), DeepGini (0.564), VanillaSM (0.563), PCS (0.562), DATIS (0.562), and Random

TABLE 5: Statistical analysis (in terms of p-value and effect size)

	Random	DeepGini	VanillaSM	PCS	Entropy	DATIS
LongTest (p-value)	6.302×10^{-17}	1.632×10^{-09}	1.249×10^{-09}	2.085×10^{-09}	1.267×10^{-09}	1.32×10^{-07}
LongTest (effect size)	6.273	2.403	2.442	2.367	2.441	1.236

TABLE 6: Average comparison results among LongTest and the compared approaches in terms of PFD

Data	Approach	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50
CancerDoc	Random	0.104	0.204	0.304	0.402	0.496
	DeepGini	0.237	0.423	0.580	0.705	0.798
	VanillaSM	0.253	0.439	0.601	0.728	0.815
	PCS	0.253	0.446	0.599	0.728	0.822
	Entropy	0.235	0.424	0.571	0.695	0.790
	DATIS	0.289	0.512	0.652	0.771	0.860
	LongTest	0.481	0.854	0.997	0.998	0.998
EURLEX57K	Random	0.097	0.196	0.297	0.399	0.501
	DeepGini	0.331	0.584	0.776	0.886	0.940
	VanillaSM	0.329	0.585	0.775	0.885	0.939
	PCS	0.328	0.585	0.775	0.884	0.939
	Entropy	0.325	0.575	0.773	0.885	0.940
	DATIS	0.414	0.685	0.833	0.911	0.948
	LongTest	0.537	0.835	0.933	0.967	0.981
FakeNews	Random	0.096	0.199	0.299	0.398	0.499
	DeepGini	0.246	0.440	0.601	0.731	0.825
	VanillaSM	0.257	0.465	0.626	0.748	0.835
	PCS	0.259	0.471	0.633	0.755	0.843
	Entropy	0.244	0.430	0.588	0.713	0.812
	DATIS	0.328	0.566	0.723	0.817	0.881
	LongTest	0.401	0.631	0.775	0.864	0.922

TABLE 7: Average comparison results among LongTest and the compared approaches in terms of NAPFD

Approach	Test Cases Selected				
	10%	20%	30%	40%	50%
Random	0.096	0.178	0.254	0.320	0.375
DeepGini	0.259	0.438	0.566	0.646	0.690
VanillaSM	0.267	0.451	0.580	0.658	0.701
PCS	0.266	0.454	0.581	0.660	0.703
Entropy	0.256	0.433	0.559	0.638	0.684
DATIS	0.301	0.512	0.624	0.712	0.756
LongTest	0.450	0.704	0.804	0.831	0.844

(0.501). These results further demonstrate that the trained LongTest models can still maintain the best effectiveness among existing methods when applied to similar-domain data.

Statistical analysis Table 5 presents the results of the statistical analysis, which aims to demonstrate the stability and statistical significance of the experimental results. We use two metrics for evaluation: p-value[51] and effect size [52]. **1) p-value:** Based on the existing study [51], if the p-value between LongTest and a compared method is less than 10^{-5}, the improvement achieved by LongTest is considered statistically significant. In Table 5, the p-values between LongTest and each test prioritization method range from 6.302×10^{-17} to 1.32×10^{-07}. All of which are less than 10^{-5}. This indicates that LongTest significantly outperforms all the test prioritization approaches with statistical significance. **2) effect size:** Based on Cohen's d [52], effect size is measured by d, where $|d| < 0.2$ indicates a "negligible" effect, $|d| < 0.5$ indicates a "small" effect, $|d| < 0.8$ indicates a "medium" effect, and values above 0.8 indicate a "large" effect. To ensure that the difference between LongTest and the compared approaches is "non-negligible," we require that $d \geq 0.2$. According to our experimental results, the effect sizes for all methods range from 1.236 to 6.273, which fall into the "large" effect category. This suggests that the improvement of LongTest over the compared test prioritization approaches is large.

TABLE 8: Effectiveness Comparison of LongTest, Dissector, and SelfChecker on DNN Models

Dataset	Approach		
	Dissector	SelfChecker	LongTest
CancerDoc	0.694	0.664	0.876
EURLEX57K	0.823	0.751	0.869
FakeNews	0.756	0.727	0.837

Effectiveness comparison (in terms of PFD) Table 6 compares LongTest with other test prioritization approaches in terms of PFD. When executing different ratios of test cases, LongTest consistently maintains the highest effectiveness across all scenarios. Notably, on the CancerDoc dataset, when executing 50% of the tests, LongTest is able to detect 99.8% of the misclassified tests. Similarly, on the EURLEX57K dataset, with 50% of the tests executed, LongTest can identify 98.1% of the misclassified tests. This indicates that LongTest outperforms all other test prioritization approaches when evaluated using the PFD metric.

Effectiveness comparison (in terms of NAPFD). Table 7 compares the effectiveness of LongTest and other test prioritization approaches using the NAPFD metric across varying percentages of selected test cases. The compared approaches include DATIS, DeepGini, VanillaSM, PCS, Entropy, and Random. We highlight the best-performing approach in each setting using gray shading. As shown in Table 7, LongTest consistently outperforms all other methods at each test execution percentage, which demonstrates that LongTest performs better than all the compared test prioritization methods when evaluated using the NAPFD metric.

Efficiency evaluation Table 9 presents the running time of LongTest and the compared test prioritization methods. The total running time of LongTest is less than 8 minutes: text embedding takes 5 minutes, contrastive learning takes 2 minutes, and prediction takes less than 1 second. The confidence-based test prioritization methods (including DeepGini, VanillaSM, PCS, and Entropy) have a running time of less than 1 second. The method DATIS takes less than 1 minute. Although LongTest is not as efficient as the other test prioritization approaches, it achieves an average effectiveness improvement ranging from 9.61% to 70.86%. Considering the trade-off between effectiveness and efficiency, LongTest remains a practical option.

Answer to RQ1: *LongTest outperforms all the compared test prioritization approaches, including DATIS, DeepGini, VanillaSM, PCS, Entropy, and Random. Compared with the non-random baselines, LongTest achieves improvements ranging from 9.61% to 16.31%.*

TABLE 9: Time cost of LongTest and the compared test prioritization approaches

Time cost	Approach						
	LongTest	Random	DeepGini	VanillaSM	PCS	Entropy	DATIS
Text Embedding	5 min	-	-	-	-	-	-
Contrastive Learning	2 min	-	-	-	-	-	-
Prediction	<1 s	<1 s	<1 s	<1 s	<1 s	<1 s	<1 min

TABLE 10: Effectiveness of LongTest and the compared approaches on BBCNews Data in terms of APFD

Approach	Random	DeepGini	VanillaSM	PCS	Entropy	DATIS	LongTest
APFD	0.501	0.564	0.563	0.562	0.568	0.562	0.591

5.2 RQ2: Impact of Number of Chunks on LongTest

Objectives: In the LongTest workflow, an input test (a long text file) is first divided into multiple small chunks. The reason for splitting the long text into chunks is that the embedding models used to convert long text into embeddings typically have input token limitations, meaning they are constrained by the length of the input text and cannot process information beyond this limit. By dividing the long text into smaller chunks, we aim to better capture information from the entire text.

However, the number of chunks can influence the quality of the generated embeddings, thereby affecting the overall effectiveness of LongTest. For instance, if the number of chunks is too small, the chunk size can exceed the input limitations of the embedding model, resulting in information loss. In this research question, we explored the impact of the number of chunks on the effectiveness of LongTest.

Experimental design: To investigate the impact of chunk number on the effectiveness of LongTest, we kept all other workflows in LongTest unchanged, adjusting only the chunk number to values of 5, 10, 15, and 20, and compared LongTest's effectiveness. We used the metrics APFD and PFD for evaluation.

Results: The experimental results for RQ2 are presented in Table 11 and Figure 3. Table 11 shows the effectiveness of LongTest when the input long text file is divided into different numbers of chunks. We see that as the number of chunks increases, the effectiveness of LongTest improves. Figure 3 visually demonstrates this relationship. Specifically, Figure 3(a) shows the evaluation using the PFD metric, where the X-axis represents the percentage of tests executed, and the Y-axis represents the effectiveness of LongTest. We see that LongTest with 20 chunks (represented by a black line) achieves the highest effectiveness. As the number of chunks increases, the effectiveness of LongTest gradually rises. Figure 3(b) presents the evaluation results based on the APFD metric, with the X-axis representing the number of chunks and the Y-axis representing the APFD value of LongTest. We also see that as the number of chunks increases, the effectiveness of LongTest improves.

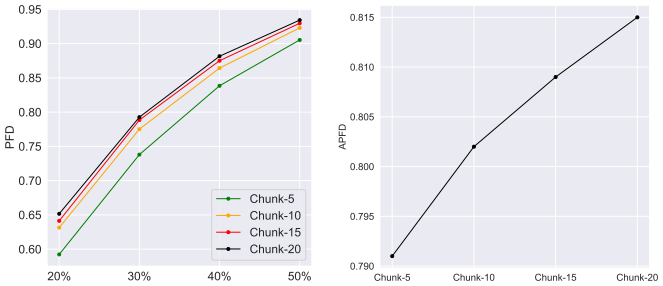
However, although increasing the number of chunks can improve the effectiveness of LongTest, the improvement effect gradually slows down. From Figure 3(a), we see that the gap between the curves for Chunk-10 and Chunk-15 is much smaller than the gap between Chunk-5 and Chunk-10, and the curve for Chunk-20 nearly overlaps with that of Chunk-15. This indicates that as the number of chunks continues to increase, the growth in effectiveness becomes limited.

TABLE 11: The PFD values of LongTest with different numbers of chunks

Chunks	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50
Chunk-5	0.372	0.592	0.738	0.838	0.905
Chunk-10	0.401	0.631	0.775	0.864	0.923
Chunk-15	0.405	0.641	0.788	0.875	0.930
Chunk-20	0.412	0.652	0.793	0.882	0.934

TABLE 12: The APFD values of LongTest with different embedding models

Embedding Model	Dataset			Average
	CancerDoc	EURLEX57K	FakeNews	
DistilRoBERTa	0.916	0.873	0.815	0.868
MPNet	0.883	0.875	0.798	0.852
MiniLM	0.871	0.883	0.795	0.849
Longformer	0.870	0.861	0.826	0.852
BigBird	0.871	0.856	0.818	0.848
E5	0.863	0.859	0.807	0.843
LaBSE	0.946	0.821	0.786	0.851
AWELD	0.806	0.824	0.746	0.792
GTR	0.902	0.847	0.774	0.841



(a) PFD (b) APFD

Fig. 3: The APFD and PFD values of LongTest with different numbers of chunks

Answer to RQ2: *When the number of chunks increases, the effectiveness of LongTest improves. However, as the number of chunks continues to increase, the growth in effectiveness becomes limited.*

5.3 RQ3: Impact of Different Embedding Models on LongTest

Objectives: Within the LongTest framework, text chunks are converted into embedding vectors, which serve as the foundation for test prioritization. Different embedding models can capture different semantic nuances, contextual relationships, and feature representations, impacting the effectiveness of LongTest. By investigating the effectiveness of Longtest with different embedding models, we aim to

TABLE 13: The PFD values of LongTest with different embedding models

Embedding Model	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50
DistilRoBERTa	0.511	0.802	0.909	0.949	0.972
MPNet	0.452	0.769	0.896	0.937	0.963
MiniLM	0.454	0.749	0.899	0.942	0.967
Longformer	0.441	0.738	0.901	0.947	0.971
BigBird	0.438	0.739	0.899	0.945	0.970
E5	0.445	0.731	0.871	0.924	0.953
LaBSE	0.544	0.746	0.854	0.918	0.951
AWELD	0.322	0.575	0.769	0.874	0.938
GTR	0.439	0.726	0.858	0.924	0.960

TABLE 14: Inference cost comparison of different embedding models

Embedding Model	DistilRoBERTa	MPNet	MiniLM	Longformer	BigBird	E5	LaBSE	AWELD	GTR
Time cost	<6 min	<9 min	<5 min	<21 min	<13 min	<8 min	<7 min	<2 min	<11 min
Size	328MB	440MB	90.9MB	595MB	513MB	438MB	1.88GB	209MB	219MB

identify the models that are more suitable for LongTest to perform test prioritization.

Experimental design: In our study, we evaluated nine popular embedding models, which are DistilRoBERTa [58], MPNet [64], MiniLM [38], Longformer [42], BigBird [84], E5 [85], LaBSE [86], AWELD [40], and GTR [87]. The core function of these models is to convert text into embeddings, which involves transforming the input text into vector representations to capture semantic information. **1) DistilRoBERTa** is a simplified version of the RoBERTa model [58]. RoBERTa is a pre-trained language model particularly skilled in capturing subtle semantics and contextual relationships within text. DistilRoBERTa achieves higher computational efficiency through distillation, maintaining strong semantic understanding capabilities while reducing resource demands. **2) MPNet** combines the advantages of BERT [88] and XLNet [89], utilizing parallel masking and positional encoding to improve the model's performance in contextual semantic understanding. **3) MiniLM** is a lightweight transformer model that leverages deep distillation techniques to strike a balance between efficient performance and embedding quality. **4) Longformer** is a transformer model designed for long documents. It replaces full self-attention with a combination of local windowed attention and task-specific global attention, reducing complexity from quadratic to linear. This enables efficient processing of sequences up to thousands of tokens, achieving strong performance on long-text tasks. **5) BigBird** is a sparse-attention transformer that scales to longer sequences by combining global, local, and random attention. It reduces the quadratic complexity of standard transformers to linear, while retaining theoretical properties like universal approximation and Turing completeness. **6) E5** is a dual-encoder model that generates text embeddings through weakly-supervised contrastive pre-training. It takes in paired inputs with task-specific prompts (e.g., "query:", "passage:"), and encodes them independently using a transformer encoder followed by mean pooling. **7) LaBSE** [86] is a multilingual sentence embedding model designed to produce language-agnostic representations across multiple languages. It is based on a BERT architecture and trained using a combination of masked language modeling and translation ranking objectives. By leveraging large-scale parallel corpora, LaBSE learns to map semantically similar sentences from different languages into a shared embedding space. **8) AWELD (Average Word Embeddings with Levy Dependency)** [40] is a lightweight embedding model that generates sentence representations by averaging pre-trained dependency-based word embeddings. Instead of using deep transformer architectures, it constructs sentence embeddings by computing the mean of individual word vectors derived from dependency-based embeddings. **9) GTR (Generalizable T5-based Retriever)** [87] is built on the encoder component of the T5-based architecture and encodes sentences and paragraphs into dense vector representations.

Specifically, in our study, we kept other workflows within LongTest unchanged, modifying only the embedding models used. We then evaluated LongTest's effectiveness with each embedding model using APFD and PFD metrics. In addition, to further investigate the efficiency trade-offs among different embedding models, we additionally evaluated the inference time and model size of each model.

Results: The results for RQ3 are presented in Table 12, Table 13 and Table 14. Table 12 shows the effectiveness of LongTest when using different embedding models (based on APFD). These embedding models include: DistilRoBERTa, MPNet, MiniLM, Longformer, BigBird, E5, LaBSE, AWELD, and GTR. From Table 12, we see that LongTest with DistilRoBERTa achieves the highest average effectiveness, with an APFD value of 0.868. This exceeds all other embedding models (APFD values ranging from 0.792 to 0.852). Table 13 uses PFD as the metric to evaluate the effectiveness of LongTest with different embedding models. The table presents that LongTest with DistilRoBERTa maintains the best effectiveness in most cases (from PFD-20 to PFD-50). To conclude, LongTest with DistilRoBERTa performs better in most cases than other embedding models (including MPNet, MiniLM, Longformer, BigBird, E5, LaBSE, AWELD, and GTR).

In the following, we provide a deeper analysis of the possible reasons why DistilRoBERTa achieves relatively better effectiveness compared to other embedding models. First, DistilRoBERTa is a distilled version of RoBERTa [90], which is pretrained on a large and diverse corpus with dynamic masking. As a result, it is able to capture richer contextual semantics and inter-sentence relationships. This capability aligns well with the objectives of LongTest, which aims to prioritize long text files containing rich information and complex structures.

In contrast, while MPNet and MiniLM are efficient in general sentence embedding tasks, they are not specifically optimized for long-document understanding. MPNet was primarily trained on sentence-level data, which could limit its ability to capture hierarchical structures in longer texts. MiniLM employs a lightweight architecture, using a simplified self-attention mechanism, and is optimized more for inference speed than for preserving nuanced semantics in longer texts. This could cause it to underperform in capturing rich semantic information over longer text files.

When considering the overall balance between performance and efficiency, MiniLM is regarded as a more practical and scalable choice. This is because: 1) MiniLM has a smaller model size [38], making it more suitable for deployment in resource-constrained environments. 2) With

its efficient architecture and reduced number of parameters, MiniLM achieves faster inference and lower memory consumption, making it well-suited for large-scale text processing tasks.

Table 14 presents the inference time and model size of each embedding model. The results show that MiniLM achieves one of the lowest inference costs among all evaluated models, requiring less than 5 minutes for inference and only 90.9MB of model size. Although AWELD also achieves relatively strong efficiency, requiring less than 2 minutes for inference, its model size reaches 209MB, which is larger than the 90.9MB required by MiniLM. In comparison, larger models such as Longformer and BigBird require substantially longer inference times (less than 21 minutes and 13 minutes, respectively) and significantly larger model sizes (595MB and 513MB).

These findings indicate that although DistilRoBERTa achieves the best effectiveness, it requires a relatively larger model size (328MB) and a higher inference cost than MiniLM. In contrast, MiniLM provides lower inference time and a much smaller model size, which makes it more suitable for practical deployment in resource-constrained environments. Specifically, MiniLM maintains competitive prioritization performance while significantly reducing inference time and has much smaller model size.

Answer to RQ3: *LongTest with the embedding model DistilRoBERTa performs better in most cases compared to other embedding models (i.e., MPNet, MiniLM, Longformer, BigBird, and E5).*

5.4 RQ4: Impact of Dimension Reduction on LongTest

Objectives: Within the LongTest framework, a crucial step is using the PCA algorithm [17] for dimensionality reduction, which involves reducing the original high-dimensional embedding vectors of long text to low-dimensional vectors. This approach aims to reduce the overall runtime and enhance the efficiency of test prioritization while preserving the features of the original data. In this research question, we investigate the impact of reducing to different dimensions on the effectiveness of LongTest. This analysis helps identify the optimal dimensionality reduction level that preserves semantic information while improving computational efficiency.

Experimental design: To investigate the impact of different dimensions on LongTest, we selected the dimensions 32, 64, 128, and 256, which are common dimensions in the literature [91]. The experiments are conducted on three long-text datasets (i.e., CancerDoc, EURLEX57K, and FakeNews) and 15 classification models (cf. Section 4.2.2). We conducted experiments on all the model-dataset combinations. Detailed information can be found in Table 1, which lists all the model-dataset combinations used to evaluate RQ4. We used APFD and PFD as evaluation metrics to assess the effectiveness of LongTest with these different dimensions. Throughout the experiment, we kept all other workflows in LongTest unchanged, adjusting only the dimensionality reduced by PCA to study its effect on LongTest. We repeated the experiment across different datasets and models to validate the stability and generalizability of the results.

TABLE 15: The APFD values of LongTest with different dimensions

Dimension Vector	Dataset			Average
	CancerDoc	EURLEX57K	FakeNews	
32	0.870	0.868	0.765	0.834
64	0.871	0.881	0.787	0.846
128	0.890	0.877	0.802	0.856
256	0.864	0.878	0.798	0.847

TABLE 16: The PFD values of LongTest with different dimensions

Dimension Vector	PFD-10	PFD-20	PFD-30	PFD-40	PFD-50
32	0.412	0.698	0.862	0.915	0.951
64	0.439	0.729	0.886	0.934	0.963
128	0.473	0.773	0.902	0.943	0.967
256	0.451	0.748	0.893	0.934	0.959

Results: The experimental results of RQ4 are presented in Table 15 and Table 16. Table 15 presents the effectiveness of LongTest with different dimensions measured by APFD. We see that LongTest with a dimension of 128 achieves the highest effectiveness (with an APFD of 0.856), while LongTest with dimensions of 256, 64, and 32 results in effectiveness values of 0.847, 0.846, and 0.834, respectively. Table 13 provides further evaluations using PFD. In Table 13, we see that at various test prioritization ratios, LongTest with a dimension of 128 consistently achieves the highest average effectiveness. These results indicate that a 128-dimensional reduction optimally preserves LongTest's effectiveness.

Below, we analyze why reducing the dimensionality of long text embeddings to 128 dimensions results in LongTest achieving relatively better effectiveness. First, after the text preprocessing steps in LongTest (i.e., splitting the long text type-test input into chunks, converting them into embeddings, and aggregating the embeddings into a comprehensive embedding), the resulting concatenated embeddings tend to be high-dimensional and redundant. The purpose of dimensionality reduction is to decrease overall runtime and reduce redundancy while preserving the essential semantic information of the long text. sdfs

If the dimensionality is reduced too much, the dimensionality reduction could overly compress the data, removing useful features in the embedding space. As a result, the downstream processes (including contrastive learning and misclassification probability estimation) operate on oversimplified representations. This could lead to a decline in the effectiveness of LongTest.

Conversely, when the dimensionality is too high, redundant features could remain, and the embedding of the long text could still be high-dimensional, which could negatively impact the learning of the contrastive learning model, leading to a decline in the effectiveness of LongTest. Experimental results demonstrate that a more suitable level of dimensionality reduction is to reduce the long text embeddings to 128 dimensions.

Answer to RQ4: *Within the LongTest framework, when utilizing the PCA algorithm to reduce the dimensionality of the input long text files, the 128-dimensional reduction optimally preserves LongTest's effectiveness.*

5.5 RQ5: Impact of Main Parameters on LongTest

Objectives: In LongTest, a Random Forest model [62] is used to predict the misclassification probability of test inputs for test prioritization. The performance of Random Forest can be influenced by several key parameters, such as `max_depth`, `min_samples_split`, and `min_samples_leaf`. Therefore, this research question aims to investigate how different parameter settings affect the effectiveness of LongTest and to identify the parameter configurations that achieve the best performance in practice.
Experimental design: We systematically vary three key parameters in the Random Forest model used by LongTest: `max_depth`, `min_samples_split`, and `min_samples_leaf`. During the experiments, we varied only one parameter at a time while keeping the remaining parameters unchanged. The effectiveness of each parameter setting is evaluated using the APFD metric across the three datasets. By visualizing the results in Figure 4, we aim to clearly illustrate the impact of different parameter settings and identify the optimal configuration.
Results: Figure 4 presents the APFD results of LongTest under different parameter settings. The results indicate that LongTest shows stable performance across a wide range of parameter configurations, with only small fluctuations in APFD values. Nevertheless, certain parameter values consistently achieve slightly better performance. For the parameter `max_depth`, setting the value to 6 yields relatively high effectiveness across all datasets. For the parameter `min_samples_split`, setting the value to 4 produces relatively high effectiveness across all datasets. For `min_samples_leaf`, the value of 3 generally provides strong effectiveness. Overall, the results indicate that LongTest is not highly sensitive to parameter variations, as the APFD values remain relatively stable across different configurations. Based on the results across datasets, the parameter configuration `max_depth`=6, `min_samples_split`=4, and `min_samples_leaf`=3 appears to provide consistently strong performance and could serve as a reasonable default setting for LongTest.

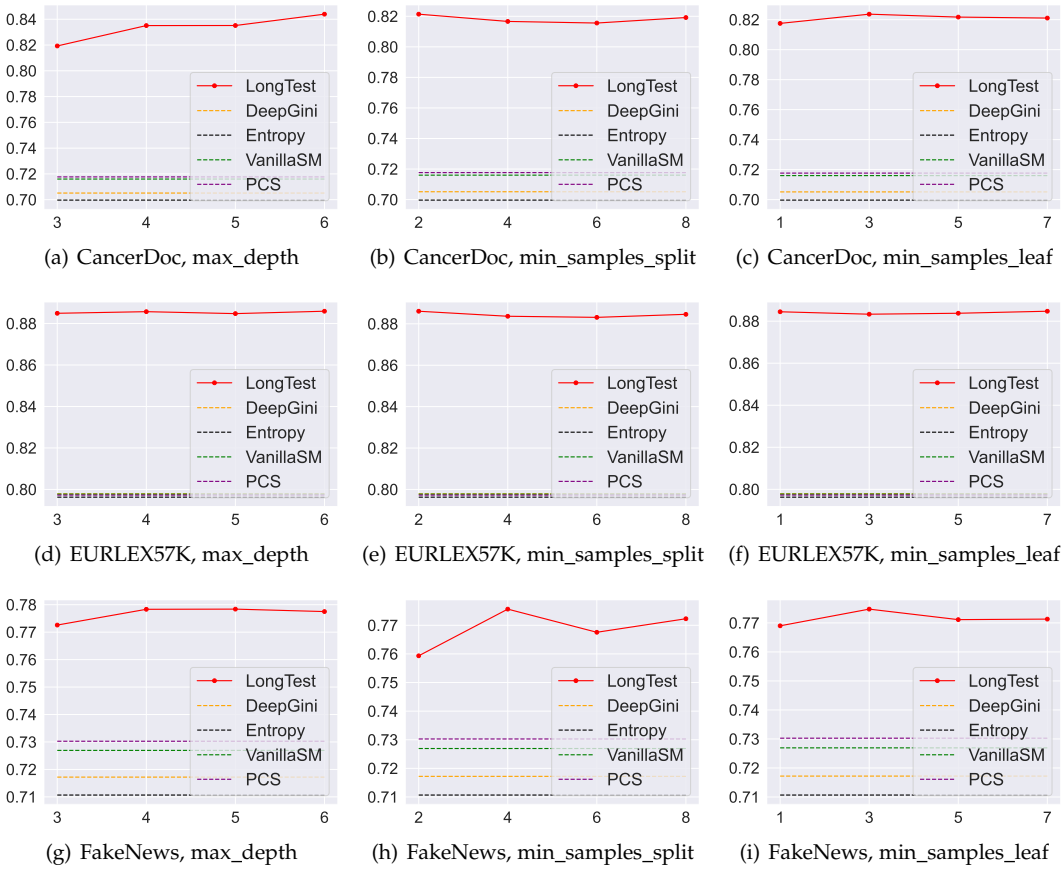
> **Answer to RQ5:** *LongTest performs stably when the values of the main parameters vary. The parameter configuration `max_depth`=6, `min_samples_split`=4, and `min_samples_leaf`=3 appears to provide consistently strong performance and could serve as a reasonable default setting for LongTest.*

5.6 RQ6: Contributions of Core Components to LongTest

Objectives: LongTest utilizes some core components to perform test prioritization, including *contrastive learning* and *PCA-based dimensionality reduction*. Specifically, within the LongTest framework, the contrastive learning model can bring the embedding vectors of misclassified tests closer to each other in the space while pushing the misclassified tests farther from the correctly classified ones. The objective is to enable better classification between misclassified and correctly classified samples. PCA-based dimensionality reduction aims to map the original high-dimensional vectors of long text files to a lower-dimensional space, thereby reducing overall runtime and improving the efficiency of test prioritization, while preserving the feature information of the original data. In this research question, we conducted an ablation study to investigate the contributions of each component to LongTest.
Experimental design: Following the methodology in the existing work [92], we conducted an ablation study to investigate the contribution of each core component of LongTest (including PCA for dimensionality reduction and contrastive learning) to its effectiveness. In the original LongTest framework, all components are included. To evaluate the impact of each individual element, we removed each component separately and compared the performance of LongTest before and after its removal. For example, to assess the contribution of contrastive learning, we compared the original LongTest with a variant that excludes the contrastive learning module.
Results: The experimental results of RQ6 are presented in Table 17 and Table 18. Table 17 shows the experimental results of the ablation study that investigates the contributions of each element to the effectiveness of LongTest. In this table, "w/o" stands for "without." For example, LongTest w/o PCA refers to the effectiveness of LongTest when the PCA component is removed. From the table, we can see that the original LongTest achieves an average effectiveness of 0.856 across all datasets, while LongTest without contrastive learning has an average effectiveness of 0.766. This indicates that removing the contrastive learning component could lead to an approximate 0.09 reduction in average APFD, demonstrating that the contrastive learning element contributes to the overall effectiveness of LongTest.

Moreover, in Table 17, we see that LongTest without PCA has an average effectiveness of 0.832, which is approximately 0.024 lower than that of the original LongTest. We also see that across different datasets, removing the PCA component reduces the effectiveness of LongTest. For example, on the EURLEX57K dataset, removing the PCA element resulted in a 0.034 decrease in LongTest's effectiveness. Although the contribution of the PCA component is relatively smaller compared to contrastive learning, its primary purpose is to reduce the runtime of LongTest by processing vectors with fewer dimensions, thereby improving overall efficiency. As demonstrated by prior research [93], [94], the PCA algorithm has been proven effective in enhancing data processing efficiency in models. Furthermore, we confirm its contribution to the efficiency of LongTest in Table 18.

In Table 18, we demonstrate how the PCA component improves the efficiency of LongTest (i.e., reduces the runtime of LongTest). The table compares the runtime of the original LongTest with that of LongTest without the PCA component. As shown in the table, the original LongTest (with PCA dimension reduction) has an average runtime of approximately 7 minutes. However, when the PCA component is removed, the average runtime increases to around 9 minutes. Most of the additional time occurs during the contrastive learning stage, with an average increase of about 2 minutes. This is because, in the original LongTest, the embedding vectors of the test (long text) are first reduced in dimensionality via PCA before being passed to the contrastive learning model. Without PCA, the high-dimensional vectors are fed directly into the contrastive learning model,



(a) CancerDoc, max_depth (b) CancerDoc, min_samples_split (c) CancerDoc, min_samples_leaf

(d) EURLEX57K, max_depth (e) EURLEX57K, min_samples_split (f) EURLEX57K, min_samples_leaf

(g) FakeNews, max_depth (h) FakeNews, min_samples_split (i) FakeNews, min_samples_leaf

Fig. 4: Impact of main parameters in LongTest

TABLE 17: Ablation study results

Approach	Dataset			Average
	CancerDoc	EURLEX57K	FakeNews	
LongTest w/o Contrastive Learning	0.811	0.801	0.687	0.766
LongTest w/o PCA	0.871	0.843	0.782	0.832
LongTest	0.889	0.877	0.802	0.856

TABLE 18: Time cost of LongTest

Time cost	Approach		
	LongTest w/o Contrastive Learning	LongTest w/o PCA	LongTest
Text Embedding	5 min	5 min	5 min
Contrastive Learning	0 min	4 min	2 min
Prediction	<1 s	<1 s	<1 s

leading to increased runtime.

Therefore, based on the experimental results, we conclude that the contribution of *dimensionality reduction* to LongTest lies in enabling it to operate more efficiently while maintaining the effectiveness.

Answer to RQ6: *The core component contrastive learning contributes to the effectiveness of LongTest, while the core component dimensionality reduction enables LongTest to operate more efficiently.*

5.7 RQ7: Impact of Different Splitting Methods on LongTest

Objectives: In the default LongTest framework, long text files are divided into fixed-size chunks for embedding. In this research question, we investigate the impact of different segmentation strategies on the effectiveness of LongTest. Specifically, we compare semantic-based segmentation, paragraph-based segmentation, and structure-aware segmentation with the default fixed-size chunking, aiming to examine which segmentation method enables LongTest to achieve better effectiveness.

Experimental design: We compared four splitting strategies (i.e., fixed-size, semantic-based, paragraph-based, and structure-aware) across the CancerDoc, EURLEX57K, and FakeNews datasets. For each configuration, we evaluated LongTest using the APFD metric, following the same experimental setup as other RQs. Below, we present the specific principles and details of these four splitting methods.

- **Fixed-size segmentation (default)** This method is the

text-splitting approach we used in the original LongTest. In this default setting, LongTest sequentially splits a long text into fixed-size chunks, each containing the same number of tokens.

- **Semantic-based segmentation** [95] This method groups sentences into chunks based on the semantics of long text files. Specifically, each sentence is first transformed into an embedding using a pre-trained SentenceTransformer model [40]. Then, we calculate the cosine similarity between each pair of consecutive sentences. If the similarity score falls below a threshold, or if adding the next sentence would cause the accumulated word count of the current chunk to exceed a predefined limit, a new chunk is created. This strategy ensures that semantically coherent neighboring sentences are kept within the same chunk while also preventing chunks from becoming too large.
- **Paragraph-based segmentation** [96] In this strategy, the long text was divided according to its natural paragraph boundaries. Consecutive sentences within the same paragraph were grouped as a single chunk.
- **Structure-aware segmentation** [97] In this segmentation method, we perform the splitting based on the structural information of the text (e.g., paragraphs, sentences and headings). Specifically, we first divide the text according to its paragraph structure and then further split it into sentences. These sentence-level units are incrementally combined into chunks, with a length constraint applied to keep each chunk manageable. In this way, the segmentation keeps the natural structure of the text while keeping the chunks coherent.

Moreover, to ensure compatibility with the input token limit of the embedding model used, we incorporate length control mechanisms into the splitting process. For methods such as semantic-based and structure-aware segmentation, we impose a predefined maximum word count when constructing each chunk (e.g., 200 words), which helps prevent exceeding the token constraint. For example, in the semantic chunking method, we progressively merge adjacent sentences based on their semantic similarity. The merging stops when the total word count of the current chunk plus the next sentence exceeds the predefined threshold (e.g., 200 words).

These strategies ensure that each chunk remains within the acceptable length for encoding while preserving semantic coherence and structural integrity. Since the fixed-size segmentation method inherently divides long texts into chunks with a predefined and uniform number of tokens, it naturally satisfies the token limit constraint of the embedding model and does not require additional length control mechanisms.

In this context, when the number of tokens in a chunk exceeds the maximum input limit of the embedding model, we recommend using the provided fixed-size, semantic-based, or structure-aware segmentation methods. These methods explicitly control the chunk size and ensure that each chunk remains within the maximum token limit of the embedding model.

For comparison, we keep all other processes unchanged and only modify the text-splitting method, in order to evaluate the impact of different text-splitting approaches on the effectiveness of LongTest.

TABLE 19: Effectiveness of LongTest under different text splitting methods

Splitting Methods	Dataset			Average
	CancerDoc	EURLEX57K	FakeNews	
Fixity	0.889	0.877	0.802	0.856
Semantic	0.890	0.857	0.761	0.836
Paragraph	0.894	0.848	0.765	0.835
Structure	0.891	0.845	0.763	0.833

Results: The experimental results of RQ7 are presented in Table 19. From the table, we see that among all four text splitting methods, the Fixity strategy achieves the highest overall effectiveness, with an average score of 0.856 across the three datasets. This is slightly higher than the Semantic (0.836), Paragraph (0.835), and Structure (0.833) methods. The difference between the best and worst performing method is only 0.023, indicating that the impact of different splitting methods on LongTest is limited.

From the perspective of datasets, we see that Fixity is the best-performing method on the datasets EURLex57K (0.877) and FakeNews (0.802). In contrast, the paragraph-based splitting method achieves the highest score on the CancerDoc dataset (0.894), slightly above Fixity (0.889), Semantic (0.890), and Structure (0.891). The maximum difference on this dataset is only 0.005. Overall, these results suggest that while Fixity provides the best average performance and is more effective on EURLex57K and FakeNews, the overall differences among splitting methods are relatively minor, showing that LongTest is robust to variations under different text splitting approaches.

> **Answer to RQ7:** *The overall effectiveness differences of LongTest among different splitting methods are relatively slight.*

5.8 RQ8: Impact of Different Loss Functions on LongTest

Objectives: In LongTest, contrastive learning is used to enhance the differentiation between misclassified and correctly classified long-text inputs. The current implementation employs a contrastive loss based on Euclidean distance (as defined in Formula 6). In this research question, we investigate the impact of different contrastive loss functions (including cosine similarity, Manhattan distance, SimSCE, and Euclidean distance) on the effectiveness of LongTest.

Experimental design: In the experiments, we compare four contrastive learning loss functions (i.e., Euclidean-based loss, cosine-based loss, Manhattan-based loss and SimCSE contrastive loss). We conduct evaluations across the CancerDoc, EURLEX57K, and FakeNews datasets. Below, we present the principles of these four loss functions.

- **Euclidean Distance (default)** Euclidean Distance is designed to pull embeddings of similar samples (i.e., both misclassified or both correctly classified) closer together in the representation space. It uses Euclidean distance as the similarity metric. This is the default loss function used in LongTest.
- **Contrastive loss with cosine similarity** [53] This loss function replaces Euclidean distance with cosine similarity, which is typically used in NLP applications to

TABLE 20: Effectiveness of LongTest with different contrastive learning loss functions

Loss Function	Dataset			Average
	CancerDoc	EURLEX57K	FakeNews	
Manhattan	0.891	0.881	0.801	0.858
Simsce	0.898	0.734	0.674	0.769
Cosine	0.890	0.878	0.808	0.859
Euclidean	0.889	0.877	0.802	0.856

better capture angular relationships in the embedding space [40].

- **SimCSE contrastive loss** [98] This loss constructs positive pairs by encoding the same sentence twice with different dropout masks (unsupervised) or using entailment pairs from natural language inference datasets (supervised). Cosine similarity is used to maximize agreement between positives while minimizing similarity with in-batch negatives. This improves both alignment and uniformity of the sentence embedding space.
- **Manhattan-based contrastive loss** [44] This loss function uses the Manhattan distance (also known as L1 distance) instead of Euclidean distance to measure the similarity between embeddings.

Results: The experimental results of RQ8 are presented in Table 20. From the table, we see that among all four loss functions evaluated, the Cosine-based contrastive loss achieves the highest average performance across the three datasets, with an average APFD of 0.859. This is slightly higher than Manhattan (0.858) and Euclidean (0.856). Apart from SimSCE, which shows relatively lower effectiveness, the other three loss functions perform comparably, with the largest difference in their average scores being only 0.003. These results suggest that Manhattan, Cosine, and Euclidean losses all achieve strong performance on long-text datasets, with relatively small differences among them.

Answer to RQ8: *Cosine similarity performs best on average, while Euclidean and Manhattan losses also yield comparable results, with only very slight average difference among these methods.*

5.9 RQ9: Enhancing Model Performance with LongTest

Objectives: We investigate whether LongTest can select informative retraining samples to improve the performance of long text classification models.

Experimental design: To evaluate whether LongTest can effectively guide selective retraining for long-text classification, we design our experiments following the methodology of the existing research [82]:

- **Dataset Partitioning** We randomly divide each dataset into three subsets: an initial training set, a candidate set, and a test set, with a ratio of 4:4:2. We reserve the candidate set for retraining and keep the test set untouched throughout the entire process to ensure unbiased evaluation.
- **Initial Training** In the first round, we train a baseline classifier based on the initial training set only.
- **Retraining** We perform retraining in multiple rounds. In each round, we incorporate an additional 10% of samples from the candidate set into the training set without replacement. We determine which samples to add using different prioritization strategies, comparing LongTest against multiple test prioritization methods. After augmenting the training set with the newly selected samples, we retrain the classifier on the complete training set (i.e., original + newly added samples), ensuring equal treatment of old and new data.
- **Evaluation** We keep the test set fixed and never use it in training, thereby ensuring an unbiased measure across retraining rounds. We record the model's accuracy after retraining to compare the effectiveness of different test prioritization strategies in selecting informative retraining samples.

TABLE 21: Enhancing the accuracy of long-text classification models with prioritized tests

Approach	PFD-10	PFD-20	PFD-30	PFD-40
Random	0.848	0.851	0.865	0.874
DATIS	0.872	0.875	0.881	0.882
DeepGini	0.861	0.863	0.869	0.881
VanillaSM	0.865	0.867	0.872	0.880
PCS	0.872	0.874	0.878	0.881
Entropy	0.871	0.873	0.880	0.883
LongTest	0.875	0.880	0.884	0.887

Results: The experimental results of RQ9 are shown in Table 21, which presents the effectiveness of using different test prioritization strategies to select samples for retraining long-text classification models. From the table, we see that all six test prioritization strategies (i.e., DATIS, DeepGini, VanillaSM, PCS, Entropy, and LongTest) outperform the Random baseline across all PFD thresholds (PFD-10 to PFD-40), indicating that test prioritization methods can enhance the retraining process of long-text classification models.

In particular, in the table, the best-performing method in each case is highlighted in gray. As shown, LongTest consistently achieves the highest accuracy, with scores of 0.875, 0.880, 0.884, and 0.887 at PFD-10, PFD-20, PFD-30, and PFD-40, respectively. This suggests that LongTest is more effective than other test prioritization strategies in selecting samples for retraining.

Answer to RQ9: *Existing test prioritization strategies can effectively enhance the retraining of long-text classification models, with LongTest demonstrating the best performance among all evaluated methods.*

5.10 RQ10: Impact of Different Ranking Models on LongTest

Objectives: We investigate the impact of different ranking models on the effectiveness of LongTest, aiming to identify the most suitable ranking model.

Experimental design: To investigate the impact of different ranking models on the effectiveness of LongTest, we generated four variants of LongTest incorporating different ranking models. The default LongTest employs Random Forest (RF) [99] as the ranking model. The only difference among these variants lies in the ranking model, while all other procedures remain identical to those in the original LongTest framework. The adopted ranking models used for comparison include Decision Tree (DT) [60], Logistic

TABLE 22: Effectiveness of LongTest with Different Ranking Models

Ranking Model	Dataset			Average
	CancerDoc	EURLEX57K	FakeNews	
DT	0.872	0.747	0.675	0.764
LR	0.847	0.865	0.785	0.832
KNN	0.862	0.840	0.746	0.816
XGB	0.882	0.868	0.786	0.845
RF	0.889	0.877	0.802	0.856

Regression (LR) [100], K-Nearest Neighbor (KNN) [101] and XGBoost (XGB) [16]. Below, we briefly introduce these ranking models.

- **Decision Tree** constructs a tree-like model for decision making based on feature values. It recursively partitions the data space and selects the most informative features to form interpretable classification rules.
- **K-Nearest Neighbors** is a non-parametric learning algorithm that classifies instances based on the labels of their nearest neighbors in the feature space. It makes predictions by measuring similarity between samples and aggregating the outcomes of the closest data points.
- **XGBoost** builds an ensemble of decision trees in a sequential manner. It incorporates regularization and optimized training procedures to improve predictive performance and computational efficiency.
- **Logistic Regression** is a widely used linear classification model that estimates the probability of class membership using the logistic function. It performs well on cases where the relationship between features and labels is approximately linear.
- **Random Forest (default)** is an ensemble learning method that constructs multiple decision trees and aggregates their predictions to produce the final result.

Results: Table 22 presents the effectiveness of LongTest when using different ranking models. Overall, Random Forest (RF) achieves the best performance across all datasets. Specifically, RF obtains APFD scores of 0.889, 0.877, and 0.802 on CancerDoc, EURLEX57K, and FakeNews, respectively, which are consistently higher than those achieved by the other ranking models. In terms of the average performance across the three datasets, RF achieves the highest score (0.856), outperforming XGBoost (0.845), Logistic Regression (0.832), KNN (0.816), and Decision Tree (0.764).

These results indicate that Random Forest is more suitable for LongTest to perform test prioritization compared to the other evaluated models. Moreover, according to existing studies [102], Random Forest has been shown to achieve better efficiency. Therefore, in the default setting, we adopt Random Forest as the ranking model for LongTest.

Answer to RQ10: *Among the evaluated ranking models, Random Forest is shown to be the most effective for LongTest to perform test prioritization (compared with Decision Tree, Logistic Regression, K-Nearest Neighbors, and XGBoost).*

5.11 RQ11: Impact of Document Length on the Effectiveness of LongTest

Objectives: We aim to investigate the impact of document length on the effectiveness of LongTest by comparing its performance on long-text and short-text documents across different datasets.

TABLE 23: Effectiveness comparison among LongTest and baseline approaches on long and short texts in terms of APFD values

Text Type	Approach	Dataset			Average
		CancerDoc	EURLEX57K	FakeNews	
Long Text	Random	0.497	0.501	0.498	0.499
	DeepGini	0.699	0.797	0.718	0.738
	VanillaSM	0.708	0.797	0.728	0.744
	PCS	0.709	0.796	0.732	0.745
	Entropy	0.694	0.795	0.711	0.733
	DATIS	0.766	0.805	0.751	0.774
	LongTest	0.895	0.887	0.817	0.866
Short Text	Random	0.501	0.503	0.499	0.501
	DeepGini	0.701	0.798	0.726	0.741
	VanillaSM	0.709	0.798	0.738	0.748
	PCS	0.711	0.798	0.749	0.753
	Entropy	0.698	0.805	0.721	0.741
	DATIS	0.769	0.825	0.763	0.785
	LongTest	0.884	0.877	0.806	0.855

Experimental design: To investigate the impact of document length on the effectiveness of LongTest, we divide the test documents into two groups based on their lengths, namely long-text and short-text documents. We then evaluate LongTest and the baseline approaches (Random, DeepGini, VanillaSM, PCS, Entropy, and DATIS) on these two groups separately across the three datasets. The effectiveness of each approach is measured using the APFD metric.

Results: Table 23 presents the effectiveness comparison between LongTest and baseline approaches on long-text and short-text documents across three datasets. The gray shading indicates the best-performing method in each case. The results show that LongTest consistently achieves the highest APFD values in both settings. For long-text documents, LongTest obtains APFD values of 0.895, 0.887, and 0.817 on CancerDoc, EURLEX57K, and FakeNews, respectively, with an average of 0.866, outperforming all baseline approaches (APFD values range from 0.499 to 0.774). Similarly, for short-text documents, LongTest also achieves the best performance with APFD values of 0.884, 0.877, and 0.806 on the three datasets, yielding an average of 0.855. In contrast, the average APFD values of the baseline approaches range from 0.501 to 0.785.

Furthermore, Table 23 shows that the performance difference between long-text and short-text settings is relatively small. The average APFD of LongTest decreases only slightly from 0.866 (long text) to 0.855 (short text), indicating that the proposed approach maintains stable effectiveness regardless of document length. These results suggest that LongTest is robust to variations in document length and performs consistently well on both long and short texts.

Answer to RQ11: *LongTest consistently achieves the best effectiveness on both long-text and short-text documents across all datasets. The small performance difference between the two settings indicates that LongTest is robust to variations in document length.*

5.12 RQ12: Impact of Target Model Accuracy on LongTest

Objectives: In practical scenarios, the classification models to be tested could exhibit different levels of predictive accuracy due to variations in model architectures, training data quality, or training configurations. These differences may influence the distribution of misclassified samples, which could potentially affect the effectiveness of test input prioritization approaches. In this research question, we investigate whether the accuracy of the target model affects the performance of LongTest. Specifically, we simulate different model accuracy levels and analyze how the prioritization effectiveness changes under these conditions.

Experimental design: To investigate the impact of the target model accuracy on the effectiveness of LongTest, we construct models with different predictive accuracy levels and evaluate how the performance of LongTest and the compared approaches varies under these settings. Specifically, we simulate some common accuracy levels: 60%, 70%, 80%, and 90%. These levels represent a wide range of performance observed in text classification systems reported in prior studies [19]. For each accuracy setting, we apply LongTest and the compared prioritization approaches (i.e., DeepGini, VanillaSM, PCS, Entropy, and DATIS) to rank the test inputs. The effectiveness of each approach is then evaluated using the Average Percentage of Faults Detected (APFD) metric. Finally, we compare the APFD results of all approaches across different accuracy levels to analyze whether the predictive accuracy of the target model influences the prioritization effectiveness of LongTest and to determine whether LongTest consistently outperforms other test prioritization approaches across different cases.

Results: Table 24 presents the APFD results of LongTest and the compared approaches under different target model accuracy levels (60%, 70%, 80%, and 90%). The table presents that LongTest consistently achieves the highest APFD values across all accuracy levels. Specifically, LongTest obtains APFD values of 0.821, 0.841, 0.896, and 0.909 when the target model accuracy is 60%, 70%, 80%, and 90%, respectively. In comparison, the best-performing baseline method (DATIS) achieves APFD values of 0.724, 0.753, 0.791, and 0.853 under the same settings. On average, LongTest achieves an APFD value of 0.867, while the best baseline method (DATIS) achieves 0.780, representing an improvement of approximately 11.15%. These results demonstrate that LongTest consistently outperforms all existing test prioritization approaches under different predictive accuracy levels of the target model.

Answer to RQ12: *LongTest consistently outperforms other test prioritization approaches across different target model accuracy levels. Its APFD values range from 0.821 to 0.909 when the target model accuracy varies from 60%, 70%, 80%, to 90%. Compared with the state-of-the-art approach DATIS, LongTest achieves an average improvement of approximately 11.15% across different accuracy levels.*

6 Discussion

In this section, we discuss the generality of LongTest and elaborate on the potential threats to its validity. Specifically, we explain why LongTest can be applied to a wide range of long-text classification tasks and analyze potential internal and external threats that may affect the reliability of our results.

TABLE 24: Effectiveness of LongTest under Different Target Model Accuracy Levels

Approach	Accuracy				Average APFD
	60%	70%	80%	90%	
Random	0.492	0.497	0.491	0.504	0.496
DeepGini	0.686	0.714	0.742	0.842	0.746
VanillaSM	0.697	0.715	0.741	0.842	0.748
PCS	0.704	0.713	0.741	0.833	0.747
Entropy	0.684	0.712	0.737	0.821	0.738
DATIS	0.724	0.753	0.791	0.853	0.780
LongTest	0.821	0.841	0.896	0.909	0.867

6.1 Generality of LongTest

Although our evaluation was conducted using 15 text classification models and three long text datasets (cf. Section 4.2), the proposed LongTest framework is not limited to these specific subjects. LongTest is primarily designed to address the test prioritization problem in long-text scenarios and can be applied to any text classification models. This is because the core steps of LongTest include text splitting, embedding generation, embedding concatenation, dimensionality reduction, and contrastive learning. These procedures are applicable to any classification task with long-text data. By leveraging the features of long texts, LongTest can effectively capture the entire textual information. Through its contrastive learning mechanism, LongTest enhances the ability to distinguish between misclassified and correctly classified samples, thereby improving the effectiveness of test prioritization.

Moreover, in scenarios where paragraphs could be exceptionally long and exceed the token limit of embedding models, we recommend adopting fixed-size segmentation as a complementary strategy to ensure that inputs remain within model constraints.

6.2 Extensibility of LongTest

Our proposed method, LongTest, is designed as an extensible test prioritization framework, rather than being limited to specific embedding techniques or dimensionality reduction methods. The core pipeline of LongTest (comprising text splitting, embedding generation, embedding concatenation, dimensionality reduction, contrastive learning, and misclassification probability estimation) can be flexibly adapted based on the characteristics of the target classification task and user preferences. In our experiments, we evaluated several widely used embedding models, including DistilRoBERTa, MPNet, MiniLM, Longformer, BigBird, and E5.

Regarding the choice of the ranking model for estimating misclassification probabilities in the final step of LongTest, we selected Random Forest due to its high stability, fast training time, and strong generalization capabilities across diverse datasets [65]. Compared to alternatives like neural networks, Random Forests require fewer computational resources and are less sensitive to hyperparameter tuning, which is critical for improving efficiency in large-scale test

prioritization tasks. Furthermore, as shown in our evaluation (cf. Section 5.5), Random Forest consistently maintains high accuracy and demonstrates strong stability under different parameter settings. Nonetheless, LongTest is not restricted to Random Forest and could support alternative ranking models. For instance, XGBoost and LightGBM, which are known for their high predictive power and scalability, can also be integrated into LongTest in the future. We encourage future researchers to explore these alternatives, particularly in scenarios where the potential improvements in accuracy make the additional computational cost a reasonable trade-off.

6.3 Scope of Applicable Prior Techniques

In this study, we have compared our proposed approach, LongTest, against a series of strong and representative baselines applicable to the long text classification domain. Specifically, we included DATIS [72], which, to the best of our knowledge, is the current state-of-the-art method for test prioritization in text classification. Our experimental results demonstrate that LongTest significantly outperforms DATIS. Some test prioritization methods were not included in the comparison with LongTest. Below, we explain the reasons.

First, the state-of-the-art method DATIS has been shown to outperform the method CertPri [103], so we did not include CertPri as a separate baseline in our experiments. In addition, the existing test prioritization methods DeepGD [104] and NNS [105] are primarily designed for image classification scenarios. DeepGD uses VGG16 [106] to extract image features, a step that cannot be applied to textual data. In the NNS method, the "representation learning" step uses SimCLR [45], which is specifically designed for image feature extraction and is not applicable to textual feature extraction. Therefore, we did not include these two methods in our comparison.

The test prioritization method NSS [107] is specifically designed for deep neural networks, relying heavily on the internal neuron structures and their activation behaviors. Specifically, NSS evaluates test cases by analyzing neuron sensitivity and the responses of input samples on these neurons. However, in our evaluated combo models, the upstream model (e.g., DistilRoBERTa) is a pre-trained model used solely for data processing and generating embedding vectors, serving only as part of the preprocessing pipeline. The downstream models (e.g., decision trees, random forests), on the other hand, are responsible for performing the classification tasks. Since these downstream classical ML models do not possess neuron-like structures or hierarchical activation mechanisms, they cannot support the sensitivity analysis required by NSS. Therefore, the NSS method is not applicable to these models that lack neural network structures.

Another test prioritization method, TDPR [108], is also specifically designed for deep neural networks. It relies on the training dynamics of neural networks at different stages of training, which includes continuously changing information such as the model's predicted probabilities, predicted labels, and uncertainty for test inputs at each training epoch. These dynamics are captured by saving multiple intermediate snapshots of the DNN during training, which are then used to construct "learning trajectories" for guiding sample selection. However, similar to the case with NSS, downstream models used for classification tasks (e.g., decision trees, random forests) are typically trained in a one-shot manner, without a gradual training process. Moreover, these models lack neuron-level representations and intermediate training states. As a result, it is not possible to extract the kind of training dynamics required by TDPR. Therefore, the TDPR method cannot be directly applied to our evaluated combo models.

6.4 Threats to Validity

Threats to Internal Validity. The internal threats to validity primarily stem from the implementation of our proposed LongTest framework and the test prioritization approaches used for comparison. To mitigate this threat, we implemented LongTest using widely recognized libraries, including PyTorch [77], TensorFlow [76], sklearn [78], and SentenceTransformer [79] (specific versions are detailed in Section 4.5). For the compared test prioritization methods, we utilized the original implementations provided by their authors to minimize potential biases introduced during re-implementation. Another internal threat arises from the randomness in the model training process. To mitigate this threat, we conducted a statistical study by repeating all the experiments ten times and reporting the average experimental results. Moreover, we assessed the statistical significance of the results by calculating the p-value [51] and effect size [52]. This approach helps reduce the impact of randomness on our experimental findings.

Threats to External Validity. External threats to validity in our study primarily arise from the long text datasets and text classification models utilized. To mitigate this threat, we employed 15 diverse text classification models, ensuring broad applicability. Additionally, we selected datasets from three different domains: medical (Cancer Text Documents), legal (EURLEX57K), and news articles (FakeNews), to capture a variety of long-text characteristics.

7 Related Work

7.1 Test Prioritization for Traditional Software

In traditional software testing [109], [110], [111], [112], [75], [113], [114], [70], [115], [116], [117], [118], [30], test prioritization is also an important direction to improve the efficiency of testing, which aims to prioritize all the tests to reveal software bugs as early as possible. In the literature, Jiang *et al.* [119] proposed an adaptive random test case prioritization (TCP) approach that uses test distance to arrange the order of test execution. Thomas *et al.* [110] introduced a novel static black-box TCP method that examines linguistic data within test cases, such as identifier names, comments, and string literals, to represent the test cases. This technique employs topic modeling to process this linguistic information, estimating the specific functionality each test case targets and prioritizing test cases that cover different functionalities. Lou *et al.* [120] developed a mutation-based prioritization approach targeting the sequencing of test cases during software evolution. By generating mutation faults

based on code modifications between software versions, this approach simulates realistic faults to improve prioritization relevance. The method employs statistical and probabilistic models to quantify each test case's fault-detection potential based on its effectiveness in "killing" mutants.

Chen *et al.* [121] conducted an empirical evaluation of various TCP techniques and developed a machine learning-based system to recommend the best prioritization method for specific projects by leveraging test distribution patterns. Pan *et al.* [118] conducted a systematic literature review on the application of machine learning (ML) techniques in test case prioritization. Their findings indicate that ML-based prioritization methods primarily utilize features such as execution history, coverage information, code complexity, and textual data, with supervised learning being the most commonly employed technique. More recently, Chen *et al.* [122] introduced LogTCP, a framework that enhances black-box test case prioritization by analyzing test execution logs to better capture test behaviors.

Similar to the aforementioned studies, our work also addresses the problem of test prioritization. However, instead of focusing on traditional software, our approach is specifically designed for long text classification models, targeting the prioritization of long text datasets. These unstructured natural language inputs are typically lengthy and pose challenges such as embedding limitations and feature redundancy. Our proposed method, LongTest, is specifically designed based on the unique characteristics of long text files. Section 5 demonstrates its high performance.

7.2 Active Learning in Text Classification

Active learning (AL) aims to reduce annotation costs by selecting the most informative data samples for labeling, thereby maximizing model performance with minimal supervision. AL and test prioritization share a common goal (i.e., identifying the most informative inputs to optimize resource usage), but they differ in their application stages: AL focuses on optimizing the training process, whereas test prioritization targets the testing phase by prioritizing fault-revealing test cases to execute first. The use of active learning in text classification has attracted significant attention in recent years. Jacobs *et al.* [123] explored uncertainty-based AL strategies for text classification using BERT and Monte Carlo Dropout. Their results show that while strategies like BALD outperform random sampling, heuristic methods like RET and RECS fail to improve performance, which reveals challenges in designing robust AL strategies. Siddhant and Lipton [124] found that Bayesian AL methods can significantly outperform classical uncertainty sampling, but results vary across datasets and models. Dor *et al.* [125] conducted a broad empirical study, highlighting that AL effectiveness depends heavily on class imbalance and initial label bias.

7.3 Testing Deep Learning Systems

Beyond test input prioritization, DNN testing [126], [127], [128], [129], [130], [131], [132] also includes other critical areas, such as accuracy estimation [133], [134], [135] and adequacy measurement [12], [13], [136], [137]. Accuracy estimation aims to improve the efficiency of DNN testing by selecting a representative subset of the original test set to estimate the model's accuracy on the entire dataset. In the literature, Li *et al.* [133] proposed Cross Entropy-based Sampling (CES) for accuracy estimation, which minimizes the cross-entropy between the distribution of the selected subset and the original test set to achieve a representative estimation. Chen *et al.* [134] proposed Practical Accuracy Estimation (PACE), which employs a clustering-based approach. Specifically, PACE first clusters the test inputs into different groups, then utilizes the MMD-critic algorithm [138] to select prototypes from each cluster, thus achieving a practical estimation of accuracy.

For adequacy measurement [12], [13], [136], Pei *et al.*[12] proposed neuron coverage as a metric to evaluate the extent to which a test set activates the internal logic of a DNN model. Expanding on their work, Ma *et al.*[13] proposed DeepGauge, a tool that broadens neuron coverage by introducing a more comprehensive suite of criteria to assess DNN adequacy. Additionally, Kim *et al.* [136] presented the concept of surprise adequacy, which measures test input effectiveness by analyzing the behavioral variations a DL model exhibits between the training and testing set.

Our work belongs to the category of DNN test prioritization, which, along with the aforementioned accuracy estimation approaches, falls under the broader area of test optimization [134]. These approaches both aim to optimize the testing process and enhance the efficiency of DNN testing. The key difference lies in the following: test prioritization focuses on ranking all test inputs based on their likelihood of being misclassified by the DNN, without discarding any input, whereas accuracy estimation methods aim to select a small subset of test inputs that can accurately estimate the overall accuracy of the full test set. This process involves discarding some test inputs and selecting the most representative ones.

8 Conclusion

To address the challenge of high labeling costs for long text files, we propose a novel approach called LongTest, which prioritizes test inputs that are more likely to be misclassified. LongTest leverages the unique characteristics of long text files for test prioritization by incorporating two core components: **1) Embedding Generation Mechanism.** This mechanism is specifically designed to enhance the capture of information from the entire long text file. Specifically, for a long text file, LongTest divides it into smaller chunks, converts each chunk into an embedding, and concatenates all chunk embeddings to produce a final embedding vector. LongTest then applies Principal Component Analysis (PCA) to the embedding vector, aiming to decrease its dimensionality while preserving the essential characteristics of the original data. **2) Contrastive Learning.** Contrastive learning brings the embeddings of misclassified samples closer while pushing the embeddings of misclassified and correctly classified samples further apart. This approach enables more effective differentiation between misclassified and correctly classified samples, thereby improving the effectiveness of test prioritization. We conducted an extensive evaluation of LongTest involving 45 subjects, covering three datasets and 15 DNN models. The experimental results demonstrate

the effectiveness of LongTest. Specifically, LongTest outperforms all the compared test prioritization approaches, achieving an average improvement ranging from 9.61% to 16.31%.

AVAILABILITY

Our replication package is available at

https://zenodo.org/records/14851892.

ACKNOWLEDGEMENTS

This work is supported by the Luxembourg National Research Fund (FNR), Grant number C22/IS/17426831/MeMoRIA; the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 949014); Fundamental Research Funds for the Central Universities (AE89991/478).

REFERENCES

[1] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: a comprehensive review," *ACM computing surveys (CSUR)*, vol. 54, no. 3, pp. 1–40, 2021.

[2] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, and L. He, "A survey on text classification: From traditional to deep learning," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, no. 2, pp. 1–41, 2022.

[3] X. Chen, P. Cong, and S. Lv, "A long-text classification method of chinese news based on bert and cnn," *IEEE Access*, vol. 10, pp. 34 046–34 057, 2022.

[4] S. Gao, M. Alawad, M. T. Young, J. Gounley, N. Schaefferkoetter, H. J. Yoon, X.-C. Wu, E. B. Durbin, J. Doherty, A. Stroup *et al.*, "Limitations of transformers on clinical text classification," *IEEE journal of biomedical and health informatics*, vol. 25, no. 9, pp. 3596–3607, 2021.

[5] F. Wei, H. Qin, S. Ye, and H. Zhao, "Empirical study of deep learning for text classification in legal document review," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 3317–3320.

[6] Y. Zhang, B. Jin, X. Chen, Y. Shen, Y. Zhang, Y. Meng, and J. Han, "Weakly supervised multi-label classification of full-text scientific papers," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023, pp. 3458–3469.

[7] C. Che, H. Hu, X. Zhao, S. Li, and Q. Lin, "Advancing cancer document classification with r andom forest," *Academic Journal of Science and Technology*, vol. 8, no. 1, pp. 278–280, 2023.

[8] S. Hu, F. Teng, L. Huang, J. Yan, and H. Zhang, "An explainable cnn approach for medical codes prediction from clinical text," *BMC Medical Informatics and Decision Making*, vol. 21, pp. 1–12, 2021.

[9] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. Chen, "Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 177–188.

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

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

[12] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

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

[14] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[15] R. E. Wright, "Logistic regression." 1995.

[16] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[17] N. Salem and S. Hussein, "Data dimensional reduction and principal components analysis," *Procedia Computer Science*, vol. 163, pp. 292–299, 2019.

[18] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining deep neural networks and beyond: A review of methods and applications," *Proceedings of the IEEE*, vol. 109, no. 3, pp. 247–278, 2021.

[19] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[20] E. Haddi, X. Liu, and Y. Shi, "The role of text pre-processing in sentiment analysis," *Procedia computer science*, vol. 17, pp. 26–32, 2013.

[21] S. Sharmin and Z. Zaman, "Spam detection in social media employing machine learning tool for text mining," in *2017 13th international conference on signal-image technology & internet-based systems (SITIS)*. IEEE, 2017, pp. 137–142.

[22] N. Jindal and B. Liu, "Review spam detection," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 1189–1190.

[23] X. Dong, L. Qian, Y. Guan, L. Huang, Q. Yu, and J. Yang, "A multiclass classification method based on deep learning for named entity recognition in electronic medical records," in *2016 New York scientific data summit (NYSDS)*. IEEE, 2016, pp. 1–10.

[24] T. Sun, W. Pian, N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "Laficmil: Rethinking large file classification from the perspective of correlated multiple instance learning," in *International Conference on Applications of Natural Language to Information Systems*. Springer, 2024, pp. 62–77.

[25] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning?" *Advances in neural information processing systems*, vol. 33, pp. 6827–6839, 2020.

[26] H. Hu, X. Wang, Y. Zhang, Q. Chen, and Q. Guan, "A comprehensive survey on contrastive learning," *Neurocomputing*, p. 128645, 2024.

[27] X. Dang, Y. Li, M. Papadakis, J. Klein, T. F. Bissyandé, and Y. Le Traon, "Test input prioritization for machine learning classifiers," *IEEE Transactions on Software Engineering*, 2024.

[28] Y. Li, X. Dang, L. Ma, J. Klein, Y. Le Traon, and T. F. Bissyandé, "Test input prioritization for 3d point clouds," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–44, 2024.

[29] Y. Li, X. Dang, L. Ma, J. Klein, and T. F. Bissyandé, "Prioritizing test cases for deep learning-based video classifiers," *Empirical Software Engineering*, vol. 29, no. 5, p. 111, 2024.

[30] C. Nguyen, P. Tonella, T. Vos, N. Condori, B. Mendelson, D. Citron, and O. Shehory, "Test prioritization based on change sensitivity: an industrial case study," *Technical Report Series/Department of Information and Computing Sciences Utrecht University*, no. UU-CS-2014-012, 2014.

[31] X. Dang, Y. Li, M. Papadakis, J. Klein, T. F. Bissyandé, and Y. L. Traon, "Graphprior: Mutation-based test input prioritization for graph neural networks," *ACM Transactions on Software Engineering and Methodology*, 2023.

[32] Y. Li, X. Dang, J. Klein, Y. Le Traon, and T. F. Bissyandé, "Pricod: Prioritizing test inputs for compressed deep neural networks," *ACM Transactions on Software Engineering and Methodology*, 2025.

[33] J. Chen, J. Wang, X. Zhang, Y. Sun, M. Kwiatkowska, J. Chen, and P. Cheng, "Fast: Boosting uncertainty-based test prioritization methods for neural networks via feature selection," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 895–906.

[34] X. Dang, Y. Li, W. Ma, Y. Guo, Q. Hu, M. Papadakis, M. Cordy, and Y. L. Traon, "Towards exploring the limitations of test selec-

tion techniques on graph neural networks: An empirical study," *Empirical Software Engineering*, vol. 29, no. 5, p. 112, 2024.

[35] Y. Li, X. Dang, W. Pian, A. Habib, J. Klein, and T. Bissyandé, "Test input prioritization for graph neural networks," *IEEE Transactions on Software Engineering*, 2024.

[36] J. Kim, J. Ju, R. Feldt, and S. Yoo, "Reducing dnn labelling cost using surprise adequacy: An industrial case study for autonomous driving," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1466–1476.

[37] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[38] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in neural information processing systems*, vol. 33, pp. 5776–5788, 2020.

[39] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[40] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, 2019, pp. 3982–3992.

[41] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California, USA: Association for Computational Linguistics, June 2016, pp. 1480–1489.

[42] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv:2004.05150*, 2020.

[43] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[44] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 2. IEEE, 2006, pp. 1735–1742.

[45] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*. PmLR, 2020, pp. 1597–1607.

[46] A. Cutler, D. R. Cutler, and J. R. Stevens, "Random forests," *Ensemble machine learning: Methods and applications*, pp. 157–175, 2012.

[47] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[48] R. Shwartz-Ziv and A. Armon, "Tabular data: Deep learning is not all you need," *Information fusion*, vol. 81, pp. 84–90, 2022.

[49] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *Advances in neural information processing systems*, vol. 35, pp. 507–520, 2022.

[50] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate automl for structured data," *arXiv preprint arXiv:2003.06505*, 2020.

[51] T. K. Kim, "T test as a parametric statistic," *Korean journal of anesthesiology*, vol. 68, no. 6, pp. 540–546, 2015.

[52] K. Kelley and K. J. Preacher, "On effect size." *Psychological methods*, vol. 17, no. 2, p. 137, 2012.

[53] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[54] J. Mueller and A. Thyagarajan, "Siamese recurrent architectures for learning sentence similarity," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[55] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, and I. Androutsopoulos, "Large-scale multi-label text classification on eu legislation," *arXiv preprint arXiv:1906.02192*, 2019.

[56] B. Jikadara, "Fake News Detection," 2023, accessed: 2024-09-07. [Online]. Available: https://www.kaggle.com/datasets/bhavikjikadara/fake-news-detection

[57] H. Y. Koh, J. Ju, M. Liu, and S. Pan, "An empirical survey on long document summarization: Datasets, models, and metrics," *ACM computing surveys*, vol. 55, no. 8, pp. 1–35, 2022.

[58] V. Sanh, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[59] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[60] B. De Ville, "Decision trees," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 5, no. 6, pp. 448–455, 2013.

[61] J. M. Hilbe, "Logistic regression." *International encyclopedia of statistical science*, vol. 1, pp. 15–32, 2011.

[62] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[63] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 8, 2021, pp. 6679–6687.

[64] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu, "Mpnet: Masked and permuted pre-training for language understanding," *Advances in neural information processing systems*, vol. 33, pp. 16 857–16 867, 2020.

[65] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The journal of machine learning research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[66] W. Abdeen, M. Unterkalmsteiner, K. Wnuk, A. Ferrari, and P. Chatzipetrou, "Language models to support multi-label classification of industrial data," in *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2025, pp. 45–55.

[67] M. T. Colangelo, M. Meleti, S. Guizzardi, E. Calciolari, and C. Galli, "A comparative analysis of sentence transformer models for automated journal recommendation using pubmed metadata," *Big Data and Cognitive Computing*, vol. 9, no. 3, p. 67, 2025.

[68] A. Shmuel, O. Glickman, and T. Lazebnik, "A comprehensive benchmark of machine and deep learning across diverse tabular datasets," *arXiv preprint arXiv:2408.14817*, 2024.

[69] M. S. Khan, "Comparative efficiency analysis of lightweight transformer models: A multi-domain empirical benchmark for enterprise nlp deployment," *arXiv preprint arXiv:2601.00444*, 2026.

[70] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software testing, verification and reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[71] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT international symposium on software testing and analysis*, 2017, pp. 12–22.

[72] Z. Li, Z. Xu, R. Ji, M. Pan, T. Zhang, L. Wang, and X. Li, "Distance-aware test input selection for deep neural networks," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 248–260.

[73] H. Wang, J. Xu, C. Xu, X. Ma, and J. Lu, "Dissector: Input validation for deep learning applications by crossing-layer dissection," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 727–738.

[74] Y. Xiao, I. Beschastnikh, D. S. Rosenblum, C. Sun, S. Elbaum, Y. Lin, and J. S. Dong, "Self-checking deep neural networks in deployment," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 372–384.

[75] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE transactions on software engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[76] M. Abadi, "Tensorflow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN international conference on functional programming*, 2016, pp. 1–1.

[77] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot,

and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[79] N. Reimers and I. Gurevych, "Making monolingual sentence embeddings multilingual using knowledge distillation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2020. [Online]. Available: https://arxiv.org/abs/2004.09813

[80] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in artificial intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[81] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[82] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–22, 2021.

[83] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 377–384.

[84] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[85] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei, "Text embeddings by weakly-supervised contrastive pre-training," *arXiv preprint arXiv:2212.03533*, 2022.

[86] F. Feng, Y. Yang, D. Cer, N. Arivazhagan, and W. Wang, "Language-agnostic bert sentence embedding," in *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: Long papers)*, 2022, pp. 878–891.

[87] J. Ni, C. Qu, J. Lu, Z. Dai, G. H. Abrego, J. Ma, V. Zhao, Y. Luan, K. Hall, M.-W. Chang *et al.*, "Large dual encoders are generalizable retrievers," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022, pp. 9844–9855.

[88] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[89] Z. Yang, "Xlnet: Generalized autoregressive pretraining for language understanding," *arXiv preprint arXiv:1906.08237*, 2019.

[90] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[91] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[92] L. Du, "How much deep learning does neural style transfer really need? an ablation study," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 3150–3159.

[93] B. F. Ljungberg, "Dimensionality reduction for bag-of-words models: Pca vs lsa," *Semanticscholar. org*, 2019.

[94] D. Sachin *et al.*, "Dimensionality reduction and classification through pca and lda," *International journal of computer Applications*, vol. 122, no. 17, 2015.

[95] R. Qu, R. Tu, and F. Bao, "Is semantic chunking worth the computational cost?" *arXiv preprint arXiv:2410.13070*, 2024.

[96] B. Zhuo, M. Murata, and Q. Ma, "Auxiliary loss for bert-based paragraph segmentation," *IEICE TRANSACTIONS on Information and Systems*, vol. 106, no. 1, pp. 58–67, 2023.

[97] J. Zhao, Z. Ji, P. Qi, S. Niu, B. Tang, F. Xiong *et al.*, "Meta-chunking: Learning efficient text segmentation via logical perception," 2024.

[98] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," *arXiv preprint arXiv:2104.08821*, 2021.

[99] H. A. Salman, A. Kalakech, and A. Steiti, "Random forest algorithm overview," *Babylonian Journal of Machine Learning*, vol. 2024, pp. 69–79, 2024.

[100] M. P. LaValley, "Logistic regression," *Circulation*, vol. 117, no. 18, pp. 2395–2399, 2008.

[101] Z. Zhang, "Introduction to machine learning: k-nearest neighbors," *Annals of translational medicine*, vol. 4, no. 11, p. 218, 2016.

[102] M. Fernandez-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[103] H. Zheng, J. Chen, and H. Jin, "Certpri: certifiable prioritization for deep neural networks via movement cost in feature space," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1–13.

[104] Z. Aghababaeyan, M. Abdellatif, M. Dadkhah, and L. Briand, "Deepgd: A multi-objective black-box test selection approach for deep neural networks," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 6, pp. 1–29, 2024.

[105] S. Bao, C. Sha, B. Chen, X. Peng, and W. Zhao, "In defense of simple techniques for neural network test case selection," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 501–513.

[106] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[107] D. Huang, Q. Bu, Y. Fu, Y. Qing, X. Xie, J. Chen, and H. Cui, "Neuron sensitivity-guided test case selection," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 7, pp. 1–32, 2024.

[108] J. Shen, Z. Li, M. Pan, and X. Li, "Prioritizing test inputs for dnns using training dynamics," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 1219–1231.

[109] Y. Lou, J. Chen, L. Zhang, and D. Hao, "A survey on regression test-case prioritization," in *Advances in Computers*. Elsevier, 2019, vol. 113, pp. 1–46.

[110] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Software Engineering*, vol. 19, pp. 182–212, 2014.

[111] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the eighteenth international symposium on Software testing and analysis*, 2009, pp. 201–212.

[112] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*. IEEE, 1999, pp. 179–188.

[113] P. Tonella, P. Avesani, and A. Susi, "Using the case-based ranking methodology for test case prioritization," in *2006 22nd IEEE international conference on software maintenance*. IEEE, 2006, pp. 123–133.

[114] D. Di Nardo, N. Alshahwan, L. Briand, and Y. Labiche, "Coverage-based regression test case selection, minimization and prioritization: A case study on an industrial system," *Software Testing, Verification and Reliability*, vol. 25, no. 4, pp. 371–396, 2015.

[115] M. Pezze, *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons, 2008.

[116] L. Mariani, M. Pezze, O. Riganelli, and M. Santoro, "Autoblacktest: Automatic black-box testing of interactive applications," in *2012 IEEE fifth international conference on software testing, verification and validation*. IEEE, 2012, pp. 81–90.

[117] M. Brunetto, G. Denaro, L. Mariani, and M. Pezzè, "On introducing automatic test case generation in practice: A success story and lessons learned," *Journal of Systems and Software*, vol. 176, p. 110933, 2021.

[118] R. Pan, M. Bagherzadeh, T. A. Ghaleb, and L. Briand, "Test case selection and prioritization using machine learning: a systematic literature review," *Empirical Software Engineering*, vol. 27, no. 2, p. 29, 2022.

[119] B. Jiang, Z. Zhang, W. K. Chan, and T. Tse, "Adaptive random test case prioritization," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 233–244.

[120] Y. Lou, D. Hao, and L. Zhang, "Mutation-based test-case prioritization in software evolution," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2015, pp. 46–57.

[121] J. Chen, Y. Lou, L. Zhang, J. Zhou, X. Wang, D. Hao, and L. Zhang, "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*, 2018, pp. 656–667.

[122] Z. Chen, J. Chen, W. Wang, J. Zhou, M. Wang, X. Chen, S. Zhou, and J. Wang, "Exploring better black-box test case prioritization via log analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–32, 2023.

[123] P. F. Jacobs, G. Maillette de Buy Wenniger, M. Wiering, and L. Schomaker, "Active learning for reducing labeling effort in text classification tasks," in *Benelux Conference on Artificial Intelligence*. Springer, 2021, pp. 3–29.

[124] A. Siddhant and Z. C. Lipton, "Deep bayesian active learning for natural language processing: Results of a large-scale empirical study," *arXiv preprint arXiv:1808.05697*, 2018.

[125] L. E. Dor, A. Halfon, A. Gera, E. Shnarch, L. Dankin, L. Choshen, M. Danilevsky, R. Aharonov, Y. Katz, and N. Slonim, "Active learning for bert: an empirical study," in *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, 2020, pp. 7949–7962.

[126] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 876–888.

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

[128] Z. Tahereh, V. Riccio, T. Paolo *et al.*, "Deepatash: Focused test generation for deep learning systems," in *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023.

[129] V. Riccio and P. Tonella, "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, 2023, pp. 1161–1173.

[130] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Efficient and effective feature space exploration for testing deep learning systems," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–38, 2023.

[131] M. Biagiola and P. Tonella, "Boundary state generation for testing and improvement of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2024.

[132] T. Zohdinasab, V. Riccio, and P. Tonella, "Deepatash: Focused test generation for deep learning systems," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 954–966.

[133] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü, "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*, 2019, pp. 499–509.

[134] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient deep neural network testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–35, 2020.

[135] L. Zhao, T. Zhao, Z. Lin, X. Ning, G. Dai, H. Yang, and Y. Wang, "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*, 2024, pp. 16 122–16 131.

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

[137] S. Dola, M. B. Dwyer, and M. L. Soffa, "Input distribution coverage: Measuring feature interaction adequacy in neural network testing," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–48, 2023.

[138] B. Kim, R. Khanna, and O. O. Koyejo, "Examples are not enough, learn to criticize! criticism for interpretability," *Advances in neural information processing systems*, vol. 29, 2016.